Controlo dinâmico com CAN-BUS de fresadora CNC para objectos alares de grandes dimensões

Sérgio Martins Simões Dias

Dissertação para obtenção do Grau de Mestre em

Engenharia Física Tecnológica

Júri

Presidente: Prof. Luís Filipe Moreira Mendes

Orientador: Prof. Bernardo Brotas de Carvalho

Co-Orientador: Prof. Pedro José de Almeida Bicudo

Vogais: Prof. Pedro Alexandre Rodrigues Carvalho Rosa

Prof. Luís Manuel Mendonça Alves

Novembro de 2009

Agradecimentos

Gostaria de agradecer ao meu orientador Professor Bernado Brotas de Carvalho, por todo o apoio despendido ao longo do desenvolvimento do trabalho descrito nesta tese. Ao meu Co-Orientador Professor Pedro Bicudo pelos inúmeros incentivos. Aos professores Pedro Rosa e Luís Alves por se mostrarem sempre disponíveis na clarificação de dúvidas.

Um especial agradecimento à Paula Vieira por ter estado sempre presente, mesmo nos momentos mais difíceis.

Agradeço ainda a todos os meus amigos e colegas que me ajudaram ao longo deste trabalho, em especial ao Hugo Alves.

Por fim mas não menos importante um agradecimento especial à minha família e em particular aos meus pais sem os quais certamente não estaria aqui.

E obrigado a todos aqueles que me esqueci de referir.

Resumo

O trabalho aqui apresentado, corresponde ao desenvolvimento de um controlo dinâmico de moto-redutores eléctricos associados a uma fresadora-protótipo de comando numérico computorizado (CNC). A estrutura dessa fresadora já se encontra desenvolvida e tem como objectivo maquinar objectos alares de grandes dimensões.

O controlo dinâmico desenvolvido, leva em conta três aspectos: o desenvolvimento de um software-protótipo, a que se deu o nome de DACS-OPMG, a aplicação de uma rede de comunicação CAN-bus e a programação dos controladores englobados nos conversores dos moto-redutores, de modo a poderem interpretar as mensagens transmitidas pelo software-protótipo através do CAN-bus.

Palavras Chave

CAN-bus; Fresadora CNC; Moto-redutores eléctricos; Conversores de frequência;

Abstract

The work presented in this dissertation, corresponds to the development of a dynamic control system for gearmotors used by a prototype-milling machine with computerized numerical control (CNC). The structure of this milling machine is already developed and its purpose is the shape of large aliform objects.

The developed dynamic control, takes into account three aspects: the development of a software prototype named DACS-OPMG, the application of a communication protocol using CAN-bus and programming of the controllers within the converters associated to the gearmotors, so that they can interpret the messages conveyed by the software prototype using the CAN-bus.

Key words

CAN-bus; Milling Machine CNC; Electric gearmotors; Frequency changers;

Índice

Lista de Abreviações

C

CA – Corrente Alternada.
CAD – Desenho assistido por computador (Computer Aided Design)
CAI – Inspecção assistida por computador (Computer Aided Inspection)
CAM – Fabrico assistido por computador (Computer Aided Manufacturing)
CAN – Controller Area Network.
CAPP – Planeamento de processamento assistido por computador (Computer Aided Process Planning)
CC – Corrente Contínua.
CCW – Rotação no sentido anti-horário.
CMM – Máquina de medição de Coordenadas (Coordinate Measurement Machine)
CNC – Comando Numérico Computorizado (*Computer Numerical Control*).
CSMA/DCR – Carrier Sense Multiple Access/Deterministic Collision Resolution
CW – Rotação no sentido horário.

D

DLL – Biblioteca de ligação dinâmica (Dynamic-link library)

DACS-OPMG – Dynamic Axis Control System - Orthogonal Projection Motion Guide

F

FIFO – (First In, First Out)

G

GUI – Interface gráfica do utilizador (Graphical User Interface)

I

IPOS® - Marca da [SEW] para (Positioning and sequence Control System).
ISO – Organização Internacional para Padronização (International Organization for Standardization).
IST – Instituto Superior Técnico.

N

NRZ – Non Return to Zero.

O

OSI – Interconexão de sistemas abertos (Open Systems Interconnection).

P

PC – Computador pessoal (Personal Computer).

S

SDL – (Specification and Description Language)

U

USB – Universal Serial Bus.

#

3D – Três Dimensões.

Lista de Figuras

Anexos

Lista de Tabelas

Anexos

Capítulo 1

1 Introdução

O trabalho aqui apresentado tem como objectivo desenvolver um controlo dinâmico para uma fresadora-protótipo [PT103652, 2007] de comando numérico computorizado (CNC), para objectos alares de grandes dimensões, usando para tal uma comunicação CAN/USB, de modo a controlar três moto-redutores eléctricos da marca SEW-Eurodrive [SEW]. O sistema controlado obedece às equações da mecânica não relativista, com rampas de aceleração e travagem.

O controlo dinâmico desenvolvido, leva em conta três aspectos: o desenvolvimento de um software-protótipo, a aplicação de uma rede de comunicação CAN-bus e a programação dos controladores dos conversores dos moto-redutores, de modo a poderem interpretar as mensagens transmitidas pelo software-protótipo através do CAN-bus.

O software-protótipo denominado DACS-OPMG, foi escrito em linguagem C++.NET. Este programa controla as posições e velocidades dos três moto-redutores eléctricos, de modo a definir a trajectória que a fresadora deve dar à fresa para que esta possa esculpir ou blocos de poliuretano expandido ou outros materiais de resistência mecânica similar. Após a elaboração desta dissertação, irá dar-se início à escrita do documento para a emissão de uma patente nacional sobre o sotware-protótipo DACS-OPMG.

Para se implementar a rede CAN-bus e fazer com que o PC acedesse a ela, optou-se por uma interface CAN/USB assegurada pelo produto CANUSB da LAWICELAB [CANUSB].

Os códigos desenvolvidos durante a programação dos controladores dos conversores de frequência associados aos moto-redutores, foram escritos em linguagem C. É importante realçar que cada moto-redutor inclui encoders de posição que fornecem o feedback necessário para se garantir que as trajectórias são cumpridas.

Este capítulo introdutório começa por enquadrar este trabalho no seu projecto original, o desenvolvimento de um protótipo para produção automatizada de pranchas de surf, composta por uma fresadora CNC, que se dedica à maquinagem, em três dimensões, de blocos em poliuretano expandido que constituem o núcleo de pranchas de surf.

Entretanto foi incluído no projecto um digitalizador 3D, que tem como objectivo fazer um scan de uma prancha modelo, obtendo-se assim uma digitalização desta, o que depois permite ter uma reconstrução do modelo em CAD/3D. Depois de obtido o modelo em CAD é possível, gerar as trajectórias de maquinagem num programa de fabrico assistido por computador (CAM), que depois pode ser usado como “input” para o controlo dinâmico descrito nesta dissertação.

As equipas envolvidas na estrutura, de servo-mecanismos, de scan de pranchas, de CAD e CAM, estão ligadas ao Departamento de Engenharia Mecânica do IST, e são lideradas pelos professores Pedro Rosa e Luís Alves. O financiamento, a montagem e os testes da máquina são assegurados pela pequena empresa SurfinAlentejo, sedeada na região de Sines, em particular pelo "shaper" Hélio Jorge.

Actualmente está bem presente a possibilidade de se poder aplicar o protótipo a outras áreas fora do âmbito do fabrico de pranchas de Surf. Isto é, os blocos de Poliuretano expandido, podem ser aplicados em variadas áreas, tais como no núcleo de aviões ultraligeiros, barcos ou pás de aerogeradores.

Para finalizar este capítulo irá se fazer um apanhado geral sobre a tecnologia de fresagem, apresentando-se logo de seguida uma descrição geral sobre fresadoras CNC.

1.1 O Projecto - Fresadora-protótipo

A ideia de iniciar o projecto para o desenvolvimento de um protótipo de uma fresadora CNC para a produção automatizada de pranchas de surf, teve origem em 2003, com a necessidade sempre crescente da empresa SurfinAlentejo [SurfIA] em aumentar a sua produção, tendo em vista uma possível expansão do mercado, nascendo assim uma parceria entre a SurfinAlentejo e o IST.

Actualmente encontram-se desenvolvidas e documentadas duas das componentes deste projecto: a estrutura da fresadora e um digitalizador 3D, que tem como objectivo fazer um scan de uma prancha modelo, obtendo assim uma digitalização desta, e a posterior reconstrução do modelo em CAD/3D.

A estrutura foi desenvolvida pelos agora engenheiros Ricardo Carvalho e Susana Alves (na altura ainda sobre a qualidade de alunos finalistas), orientados pelo pelos professores Pedro Rosa e Luís Alves e descrito em [Carvalho, 2004]. E o digitalizador 3D foi desenvolvido pelos agora engenheiros João Amaro e Ricardo Bettencourt (na altura ainda sobre a qualidade de alunos), orientados pelo pelos professores Pedro Rosa e Luís Alves e descrito em [Amaro, 2005].

A estrutura desenvolvida para a fresadora já tem patente emitida [PT103652, 2007] pelo Instituto Nacional da Propriedade Industrial, assim como o design [DOM756, 2007].

Esta estrutura permite, com um único aperto do bruto de maquinagem, maquinar superfícies longas, nomeadamente superfícies alares, tais como, asas para aviões, pás para geradores de energia eólica, cascos para embarcações e pranchas de surf entre outros. Logo este projecto tem mais aplicações para alem do maquinar de um objecto específico como as pranchas de surf.

1.1.1 Estrutura da fresadora para grandes objectos

Quando se iniciou o projecto [Carvalho, 2004], os objectivos traçados passavam por desenvolver uma fresadora CNC de três eixos para maquinar materiais da mesma resistência mecânica que o poliuretano, respeitando áreas de trabalho de 5 m de comprimento, 1,5 m de largura e 1 m de altura, a um custo reduzido e com uma precisão global na ordem dos milímetros (ordem de grandeza para as pranchas de surf).

Finalizado o projecto, obteve-se uma estrutura que apresenta uma grande facilidade de construção e de alteração, com centro de gravidade da máquina perto do chão, a trabalhar sobre cremalheiras dentadas dispostas como carris onde desliza um carro móvel, em posição invertida, relativamente ao convencional. Inversão esta que é caracterizada pelo sentido de baixo para cima que os servo-mecanismos e as ferramentas optam durante o processo de maquinação, o que possibilita uma zona de fresagem mais limpa de aparas [Figura 1.1].

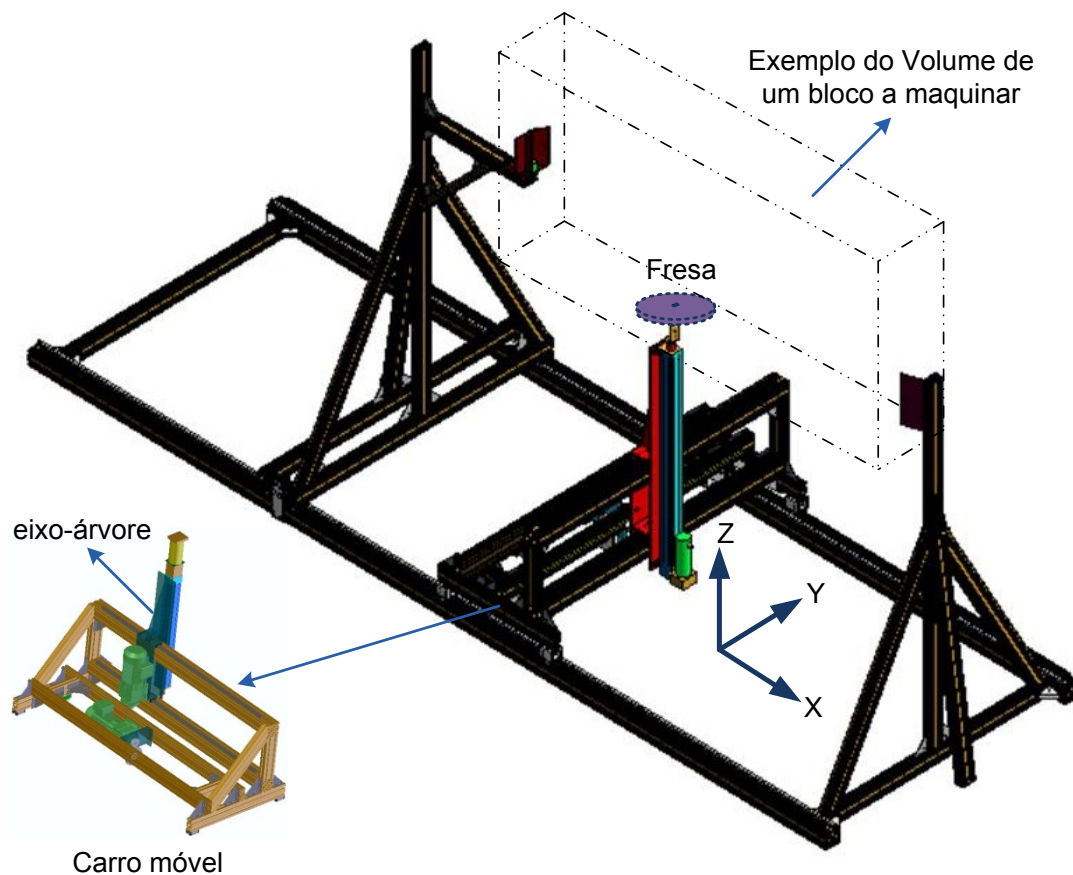



Figura 1.1: Representação da estrutura obtida para a fresadora-protótipo CNC [Carvalho, 2004].

Com o centro de gravidade da máquina perto do chão, aliado ao facto da estrutura estar presa ao chão, ajuda a criar estabilidade em termos dinâmicos, permitindo deste modo acelerações e desacelerações sem trepidações, evitando por sua vez deformações e tensões indesejáveis.

Tendo ainda em conta que o eixo-árvore pode ser móvel nas três direcções, deslocando-se assim sob a matéria-prima, e que por outro lado apresenta uma ferramenta de corte suficientemente larga para trabalhar a superfície dos dois lados, com um único aperto, conclui-se então que não é necessário o reposicionamento do bruto de maquinagem.

Em termos de comprimento, pode comportar objectos maiores que o seu actual comprimento (5.3 m), isto por que é ajustável ao tamanho da peça a maquinar, por adição de "módulos de carril", caracterizando assim esta estrutura como sendo modular.

Como já foi dito, esta estrutura foi estudada e desenvolvida para maquinar num só acto superfícies de grandes dimensões, nomeadamente superfícies longas.

No que diz respeito ao custo dos materiais envolvidos nesta estrutura e o seu peso, o trabalho desenvolvido por [Carvalho, 2004] concluiu que a solução final encontrada, em relação ao inicialmente proposto, apresenta uma série de vantagens que tornam a máquina um produto inovador e com potencial para ter um baixo custo, para o seu mercado alvo.

Assim resumindo, esta fresadora tem uma analogia com um carro que se desloque sobre carris, com a diferença que os carris são substituídos por cremalheiras, dando um aspecto de carris dentados em que as rodas são também dentadas de forma a que o carro tenha sempre uma posição bem precisa. Isto corresponde a direcção (eixo dos *xx*). Nas direcções perpendiculares o mesmo princípio repete-se, para que a fresa se possa mover em todas as direcções possíveis [Figura 1.2].

Figura 1.2: Fotografia da estrutura criada para o protótipo da fresadora CNC [Carvalho, 2004].

1.1.2 Digitalizador 3D

Este sistema pode servir de alternativa para se obter ficheiros CAM, descritos no capítulo 2, que posteriormente podem ser usados como inputs no programa apresentado no capítulo 4.

Assim, a digitalização de formas 3D consiste na geração de uma nuvem de pontos (coordenadas 3D) a partir de um modelo físico. A informação gerada durante este processo é posteriormente transferida para um sistema de reconstrução de modelos, de forma a gerar um modelo conceptual de superfícies com base nos pontos de coordenadas 3D, de onde é possível gerar trajectórias de maquinagem à posteriori num programa CAM. Obtendo-se deste modo os ficheiros CAM de uma forma mais confortável para o utilizador.

1.1.3 Material alvo – Material com resistência perto do poliuretano expandido

O controlo dinâmico desenvolvido para esta fresadora foi desenhado e desenvolvido nesta primeira fase para definir as trajectórias tridimensionais que a fresadora deve dar à ferramenta de corte, logo não contempla um quarto controlo para a velocidade de rotação da ferramenta de corte nem como deve ser orientado este corte.

Isto acontece porque para fresar o poliuretano expandido, não é necessário grandes preocupações com os parâmetros referentes ao modo de corte, basta que a ferramente de corte mantenha uma velocidade de rotação e translação estável para que possa fresar este tipo de materiais sem o estragar. Isto só é possível, porque a resistência do Poliuretano em comparação com as ferramentas de corte, é desprezável.

Mas esta é uma das vertentes a ter em conta, para a próxima fase do projecto – montagem e adaptação às condições reais de fresagem.

1.2 Tecnologia de fresagem

O corte por arranque de apara é um processo tecnológico de alteração de forma, que, através da remoção de material, sob a forma de aparas, permite a obtenção de um componente de geometria determinada. Assim sendo, pode-se definir a fresagem como sendo a operação de corte por arranque de apara destinada à produção de superfícies planas ou curvas utilizando para tal ferramentas multicortantes [Mesquita, 1997].

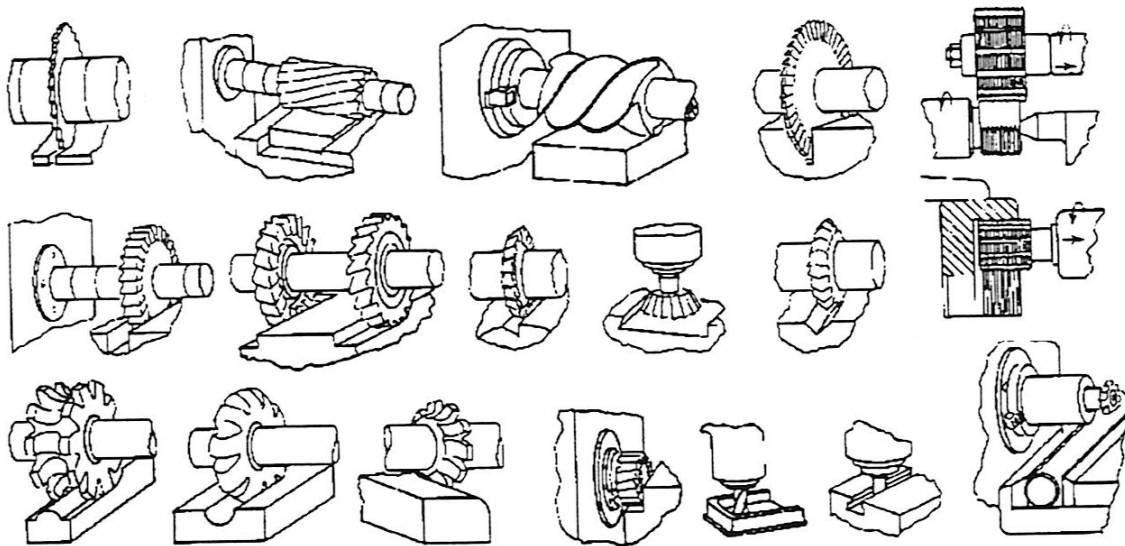



Figura 1.3: Exemplos de várias operações de fresagem [Mesquita, 1997]

Para simplificar pode-se enquadrar o processo de fresagem dentro de dois tipos, sendo estes, fresagem tangencial ou fresagem frontal [Youssef, 2008].

1.2.1 Fresagem tangencial

Numa fresagem tangencial ou cilíndrica de corte periférico, as arestas de corte da fresa poderão ser paralelas ou oblíquas em relação ao eixo de rotação da ferramenta. Caso seja oblíqua, esse ângulo de obliquidade é designado por ângulo da hélice [Figura 1.4]

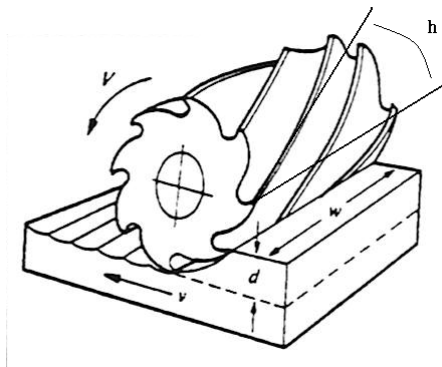



Figura 1.4: Representação da criação de uma superfície plana utilizando uma fresa cilíndrica de corte periférico em que h=ângulo da hélice [Mesquita, 1997]

Este tipo de fresagem é geralmente usado quando temos fresadoras horizontais, em consequência disso este processo é muitas vezes apelidado de fresagem horizontal em alguma literatura. Este tipo de fresagem produz uma apara de espessura variável (antes de ser cortada), sendo a sua largura medida paralelamente ao eixo de ferramenta.

Ainda no caso de fresas de corte periférico de geometria igual a da Figura 1.4, elas podem cortar em dois sentidos. No sentido tradicional, contrário ao do avanço, neste caso os dentes da

fresa penetram na peça no sentido contrário ao da aproximação desta. Ou então no outro sentido, o mesmo que o do avanço da peça.

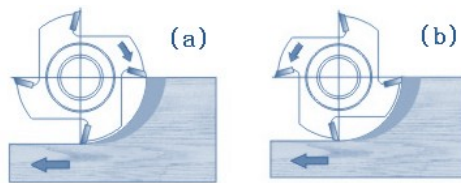



Figura 1.5: Corte no sentido do avanço (a) e no sentido contrário (b)

1.2.1.1 Corte no sentido contrário ao do avanço da peça (tradicional)

Quando a ferramenta corta no sentido contrário ao do de avanço da peça a maquinar, a apara é cortada segundo a secção variável: começa com uma espessura perto do zero e termina com uma espessura máxima, fazendo com que as folgas sejam anuladas durante o processo de corte, eliminado assim ao máximo vibrações.

Neste caso, como foi dito, a espessura cresce desde zero até ao valor máximo; como há uma espessura mínima que a ferramenta pode cortar, abaixo dessa espessura a ferramenta não efectua um corte, mas sim um polimento por roçamento, e que faz com que haja um aquecimento anormal dos dentes e um embotamento prematuro.

1.2.1.2 Corte no sentido do avanço da peça

Neste caso em que a ferramenta corta no sentido do avanço, a apara também é cortada segundo a secção variável, começando com uma espessura determinada e terminando com uma espessura nula. Assim, para o mesmo passo, a apara é mais grossa e mais curta e, portanto, as forças de corte são maiores podendo deste modo criar vibrações.

Como qualquer tipo de vibrações/oscilações são prejudiciais e podem dar origem à ruptura da fresa, o corte no sentido do avanço só se pode fazer quando as máquinas têm dispositivos especiais para anular folgas.

O corte, quando feito no sentido do avanço, começa por ser efectuado com uma secção determinada da apara, o que origina um choque no início do corte, de que pode resultar na fractura dos dentes da fresa (sobretudo se são carbonetos). Nessa situação os dentes devem ter um ângulo de ataque negativo, para serem robustos e o material do dente deve ser mais dúctil.

1.2.2 Fresagem frontal

Este processo é usado para se obter superfícies planas perpendiculares ao eixo de rotação da ferramenta. Para tal, usa-se normalmente fresas de facejar.

A espessura da apara também é variável, sendo o seu valor máximo correspondente ao avanço por dente. A profundidade de corte e consequentemente a largura da apara são determinadas pelo ajuste axial da fresa relativamente à superfície inicial da peça.

Pode-se ainda fazer uma pequena referência para as fresas de topo que são utilizadas para produzir rasgos, caixas e contornos, e corta, tal como as fresas de facejar, simultaneamente com as arestas laterais e com as de topo.

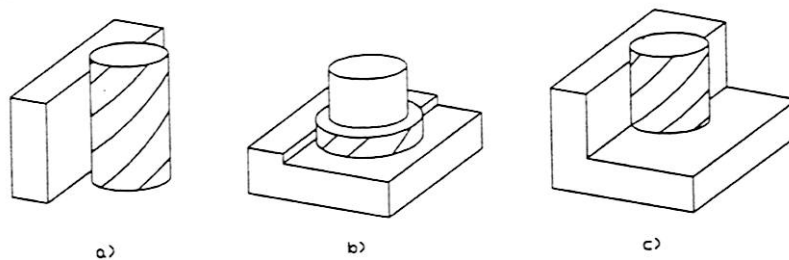



Figura 1.6: Comparação: Fresas cilíndricas de corte periférico (a), de facejar (b) e de topo (c) [Mesquita, 1997]

1.3 Fresadoras CNC

Esta secção começa por dar uma perspectiva geral do processo, isto é, quais os passos que normalmente tem que ser percorridos para maquinar uma peça numa fresadora CNC [Figura 1.7] antes de se prosseguir com a descrição dos modos constituintes de uma fresadora CNC .

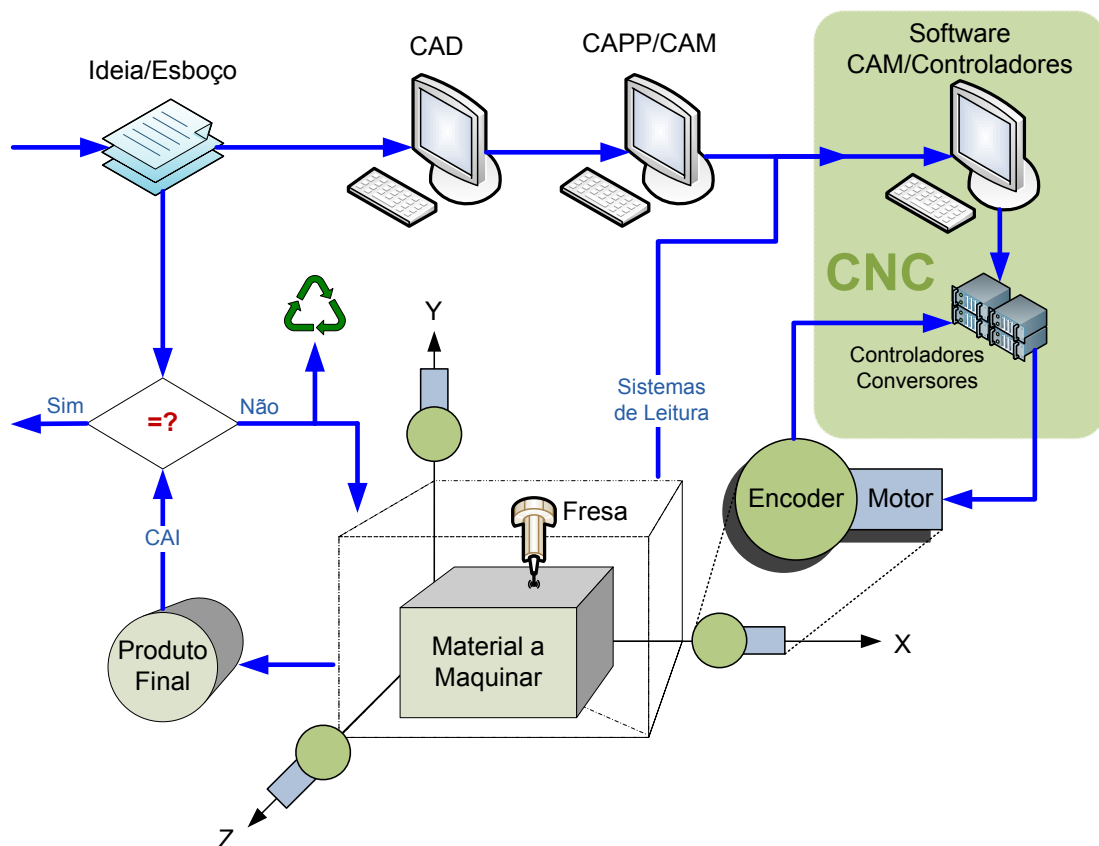



Figura 1.7: Esquema dos processos envolvidos na maquinação de uma peça envolvendo uma fresadora CNC

Ideia/Esboço – Depois de se ter tido a ideia do que se pretende obter como produto final na fresadora, deve-se por em papel na forma de um esboço. Esboço esse que deve ter em conta, se possível, onde será usado e como, de que material será feito, qual o seu tamanho, etc.

CAD – CAD (Computer Aided Design) é uma ferramenta de auxílio à confecção de desenhos, principalmente em engenharia. Assim é através do CAD que passamos a Ideia/Esboço para o computador transformando-o numa forma geométrica modelar quer em 2D ou 3D.

CAPP e CAM – CAPP (Computer Aided Process Planning), é usado para gerar informação sobre a máquina-ferramenta e o material a maquinar, como por exemplo o tipo de fresa a usar, condições de corte a cumprir, etc. Depois de obtido o CAPP passamos para o CAM (Computer Aided Manufacturing), onde será guardado então as instruções para o movimento da máquina-ferramenta com base na geometria criada pelo CAD tendo em conta as imposições de maquinaria registadas no CAPP.

CNC – É no Comando Numérico Computorizado que se interpreta o ficheiro CAM para posteriormente ser convertido em instruções de posição e velocidade para os servo-motores através dos controladores dos servos.

Acção de remoção de material – Conforme os servos recebem as instruções transformam-nas em movimentos ao logo de eixos translacionais/rotacionais criando assim todo o processo de fresagem. Normalmente o estado da Fresadora assim como o processo de fresagem é monitorizado e o feed-back enviado para ser analisado.

CAI – CAI (Computer Aided Inspection) é usado para no fim do processo para inspecionar a peça obtida e comparar com os requisitos imposto pela ideia/esboço criado no início de todo este processo, em que para tal se usa normalmente o CMM (Coordinate Measurement Machine) que faz uma comparação criando um sistema de coordenadas da peça final em CAD de forma a poder comparar com o CAD criado.

No que diz respeito à fresadora em si, esta é normalmente constituída por um corpo estrutural, um carro sobre o qual se encontra uma mesa onde se coloca o material a maquinar e um cabeçote que acopla as ferramentas de corte. Normalmente a mesa pode-se mover numa direcção (chamado movimento longitudinal) e por estar montada sobre o carro que lhe dá um movimento perpendicular ao primeiro (chamado movimento transversal).

Como vantagens, este tipo de máquinas apresentam uma elevada eficiência, produzem superfícies com bom acabamento e permitem uma elevada flexibilidade em gerar formas.

Do ponto de vista dos seus componentes uma máquina-ferramenta CNC pode ser constituída por estrutura, servomecanismos, sistemas de leitura e controladores.

1.3.1 Estrutura

A Estrutura de uma máquina-ferramenta deve apresentar algumas características fundamentais, como uma boa relação entre rigidez e alto peso, grande capacidade de absorção de vibrações, servos dos eixos de elevado binário e um bom sistema de arrefecimento [Carvalho, 2004].

As máquinas fresadoras são geralmente classificadas de acordo com a posição do seu eixo-árvore em relação à mesa de trabalho, sendo o eixo-árvore a parte da máquina onde se fixa a ferramenta e a mesa de trabalho onde se fixa a peça a maquinar. Assim, as fresadoras podem ser horizontais e/ou verticais.

1.3.1.1 Fresadoras horizontais

As fresadoras horizontais são caracterizadas por possuírem o eixo-árvore em posição horizontal [Figura 1.8], recebendo o movimento de rotação através deste. O eixo-árvore é suportado numa das extremidades pelo braço de apoio de modo a aumentar a rigidez do sistema. A mesa, de movimentos cruzados (longitudinal e transversal), é apoiada na consola, que por sua vez se desloca verticalmente. Se a dimensão e a massa da peça (e por consequência da mesa) for muito grande, o apoio em consola não será adequado. Nessa situação utilizam-se fresadoras de mesa ou bancada fixa [Mesquita, 1997].

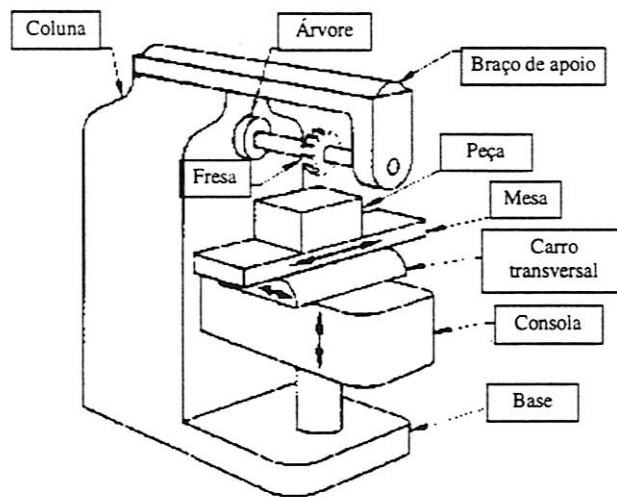



Figura 1.8: Fresadora horizontal em consola [Mesquita, 1997]

1.3.1.2 Fresadoras verticais

As fresadoras verticais, de mesa móvel ou fixa, são caracterizadas pela posição vertical da árvore de trabalho. A nomenclatura dos principais órgãos de uma fresadora vertical de mesa móvel é apresentada na Figura 1.9. Para além dos movimentos cruzados da peça, promovidos pelos carros longitudinal e transversal, a consola pode-se deslocar verticalmente, quer para o posicionamento,

quer para avanço de trabalho. Também a manga da árvore se pode deslocar na direcção vertical [Mesquita, 1997].

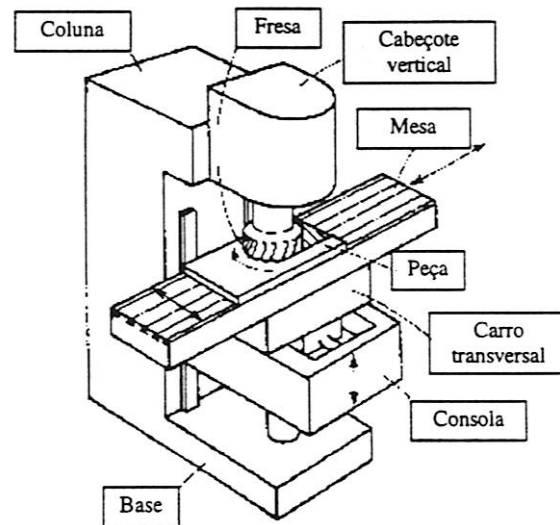



Figura 1.9: Fresadora vertical em consola [Mesquita, 1997]

1.3.1.3 Outras fresadoras

No topo das fresadoras que advêm das horizontais, temos a fresadora universal. Esta dispõe um eixo-árvore na horizontal em que a mesa tem a possibilidade rodar em torno do eixo vertical, facilitando assim a maquinação de materiais dos quais se pretende obter formas helicoidais

Existem outros tipos de fresadoras que advêm das horizontais e das verticais mas que possuem um diferente modo de funcionamento. Destas fresadoras mais especializadas, pode-se destacar, então, a fresadora copiadora e a pantográfica. A fresadora copiadora trabalha com uma mesa e dois cabeçotes (o cabeçote apalpador e o cabeçote de maquinagem) e tem a finalidade de copiar um dado modelo. A fresadora pantográfica executa um movimento de coordenadas operado manualmente e, tal como a anterior, copia um dado modelo permitindo, no entanto, trabalhar detalhes como canais e pequenos raios que são complicados de ser obtidos numa fresadora copiadora [Carvalho, 2004].

1.3.2 Servomecanismos

Um servomecanismo é um sistema que funciona para posicionar ou controlar uma carga em resposta a um sinal de entrada que é capaz de fornecer somente uma pequena potência. O sistema funciona de modo a minimizar qualquer diferença que possa existir entre as posições reais e desejadas da carga. Estes sistemas devem apresentar algumas propriedades, tais como, baixo tempo de resposta, elevada rigidez, virtualmente sem folgas e não aquecerem em demasia [Carvalho, 2004].

Dependendo das aplicações em particular, os servomecanismos podem ser, electro-mecânicos, electro-hidráulicos, electro-pneumáticos, electro-magnéticos, hidráulicos e pneumáticos.

Para este projecto, o estudo desenvolvido por [Carvalho, 2004], chegou à conclusão que de todos os servomecanismos acima mencionados, aqueles que teriam mais interesse para a aplicação numa fresadora CNC, com os requisitos impostos neste projecto, são os servomecanismos electro-mecânicos.

Os servomecanismos electro-mecânicos são constituídos por uma parte de comando electrónica, que utiliza o redutor de erros (servomotor CA ou CC) para actuar directamente no sistema de transmissão de movimento mecânico, como por exemplo fusos, correias, cremalheiras, etc. Pode-se ainda realçar que os servomecanismos escolhidos possuem encoders, os quais são descritos na próxima sub-secção.

1.3.2.1 Encoders

Encoders são transdutores de movimento capazes de converter movimentos angulares ou lineares em impulsos digitais eléctricos, e posteriormente retirar informação sobre as velocidades e posições correntes dos motores. Geralmente são acoplados no final do eixo de transmissão do motor, para que deste modo se possa obter uma informação mais precisa e segura da sua velocidade. Velocidade esta que é medida directamente por um sensor (caso seja possível), ou então é calculada pelo controlo de posição associado à rotação. O método para determinar posições passa por contar os pulsos gerados pela rotação numa unidade de tempo, detectando assim intervalos entre pulsos.

Os encoders principais podem ser classificados como sendo ópticos ou magnéticos, o que muda é somente o modo como detectam, pois os sinais de output gerados pelos dois são exactamente iguais [Suh, 2008].

Nos encoders ópticos rotativos, a leitura é feita usando um disco com janelas radiais transparentes e opacas, alternadas entre si. O disco é iluminado perpendicularmente por uma fonte de radiação electromagnética (normalmente infravermelha), que com o movimento do disco vai alternando entre janelas transparente que deixam passar a radiação ou opacas que a bloqueiam. Do outro lado do disco temos um receptor que ao absorver a radiação a converte em impulsos eléctricos [Figura 1.10]. Se a uma linha de janelas chamarmos canal, então existem ocasionalmete encoders com mais que um canal, normalmente desfasados entre eles, tal permite por exemplo determinar o sentido de rotação do disco.

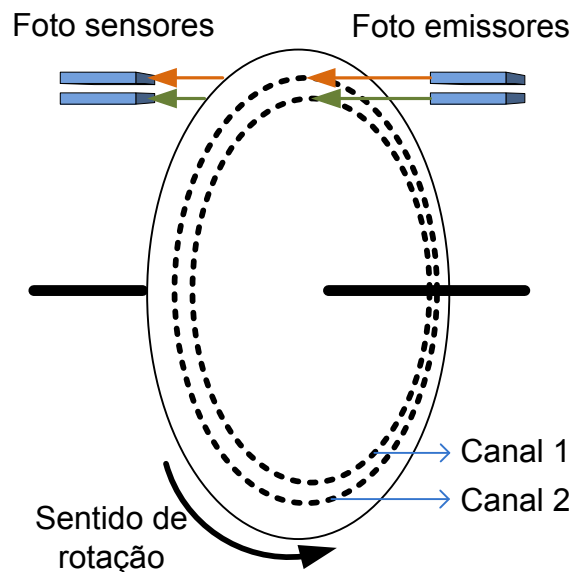



Figura 1.10: Esquema de um encoder óptico rotativo com 2 canais

Os encoders podem ser divididos em encoders incrementais e absolutos. Ambos apresentam o mesmo modo de funcionamento descrito anteriormente, a grande diferença é que os absolutos determinam uma posição bem determina e fixa, enquanto que os incrementais se referem a uma posição resultante do incremento a uma posição assumida pela mesa.

1.3.3 Sistemas de leitura

Os Sistemas de Leitura são caracterizados normalmente pelos transdutores, que são sistemas que permitem converter variações de uma unidade física numa outra unidade física. Assim, por exemplo numa definição mais específica (e bastante utilizada) é de que transdutor é um dispositivo que transforma um tipo de energia em outro, utilizando para isso um elemento sensor. Por exemplo, o sensor pode traduzir informação não eléctrica (velocidade, posição, temperatura, pH) em informação eléctrica (corrente, tensão, resistência).

Nas máquinas CNC existem dispositivos de medição ligados aos accionamentos que permitem a determinação da posição de cada um dos carros móveis [Figura 1.11]. Ajudando deste modo a determinar o ponto-zero da máquina-ferramenta CNC com base na origem das diferentes escalas de cada eixo [Carvalho, 2004].

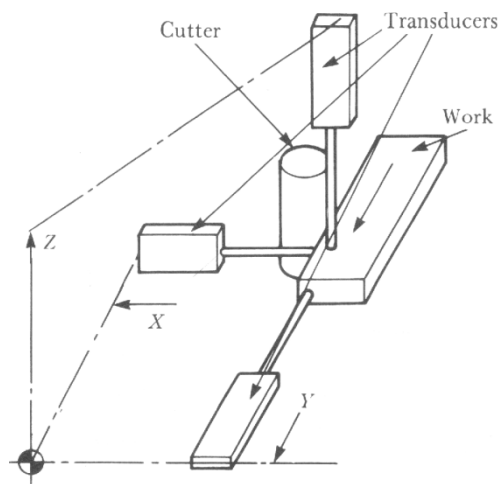



Figura 1.11: Representação do funcionamento de um sistema de leitura de coordenadas

Assim sendo, estes sistemas são utilizados nas máquinas CNC com objectivos diferentes, que podem passar por monitorizar posição dos carros móveis até níveis de potência associada. Ou ainda, para medir velocidades, temperaturas, desgastes de ferramentas, pressões de óleo nos circuitos hidráulicos, etc. Para terminar, pode-se ainda referir que um dos objectivos principais passa também pela segurança, ou seja, sensores que ao detectar uma obstrução enviam um sinal de alerta, que permite desligar a maquinaria em caso de urgência, ou mal funcionamento.

1.3.4 Controladores

No caso das máquinas CNC esta unidade consiste maioritariamente num conjunto de circuitos electrónicos, memorias e microprocessadores que lêem, interpretam e manipulam o programa de maquinagem, convertendo-o em acções mecânicas que possam ser interpretadas pela máquina-ferramenta.

1.3.4.1 Comando Numérico Computorizado (CNC)

O Comando Numérico é um sistema que interpreta dados alfanuméricos codificados, criados previamente para o efeito, de modo a controlar as acções mecânicas de uma máquina-ferramenta. Estes dados representam posições relativas entre a ferramenta e a peça de trabalho assim como outras instruções necessárias para operar a máquina. O termo Comando Numérico Computorizado (CNC) é usado quando o sistema usa um computador ou um microprocessador interno que contem memória, que possibilita deste modo guardar as rotinas. Assim no CNC as funções lógicas ficam guardadas como instruções de software em vez de usarmos hardware (circuitos lógicos) para este efeito como acontecia antes com o Comando Numérico.

Capítulo 2

2 Descrição da tecnologia usada

O estudo desenvolvido por [Carvalho, 2004], referido no primeiro capítulo, considera com base em factores de desempenho e económicos, que os servomecanismos electro-mecânicos são a melhor escolha para aplicar na estrutura desenvolvida. Tendo isto em conta, os responsáveis do lado do Departamento de Engenharia Mecânica do IST, liderados pelos professores Pedro Rosa e Luís Alves, optaram por motores-redutores assíncronos AC da marca SEW-EURODRIVE LDA [SEW] equipados com encoders e conversores de frequência com controlo. Para estabelecer a comunicação entre os moto-redutores e o PC, optou-se pelo protocolo de comunicação CAN devido à sua simplicidade de implementação. Enquanto que para fazer a ligação entre a rede CAN e o PC, optou-se por um dongle CAN/USB de baixo custo, denominado CANUSB da marca LAWICEL AB [CANUSB].

2.1 Moto-redutores com encoders SEW-EURODRIVE

Olhando para a estrutura representada na [Figura 1.1], conclui-se que a solução mais segura passa por escolher motores mais potentes para os eixos do X e Y, e um mais leve e com menos potencia para o eixo dos Z. Isto porque os dois primeiros suportam mais peso, enquanto que o que se encontra no eixo-árvore (eixo dos Z) apenas tem de suportar o próprio eixo e o motor associado à fresa, e claro convêm que este seja o mais leve possível para não aplicar mais peso no carrinho móvel.

Assim sendo, escolheram-se os motores com as seguintes designações:

Para o eixo dos X e Y temos:

Moto-Redutor 1: RF17DT80N4/BMG/ES1S

Moto-Redutor 2: R17DT80N4/BMG/ES1S

E para o eixo dos Z temos:

Moto-Redutor 3: WAF20DR63L4/BR/EH1S

A escolha das referências de cada moto-redutor foi efectuada de modo a ser mais fácil compreender as características mais relevantes dos moto-redutores [SEW]. redutores [SEW]. As referências são divididas em blocos (conjunto de letras e números separados por “/”).O primeiro bloco contem informação sobre o tipo de moto-redutor, no segundo o tipo de freio (travão) usado, e por fim, no último bloco o tipo de encoder que vem acoplado. Voltando ao primeiro bloco, a

referencia expressa diz respeito simultaneamente ao Redutor e ao Motor, ou seja, pode-se ainda dividir em duas referências, uma para cada um e obter informação sobre estes em separado. A título exemplificativoapresenta-se de seguida os esquemas interpretativos das referências de cada moto-redutor [Figura 2.1][Figura 2.2].

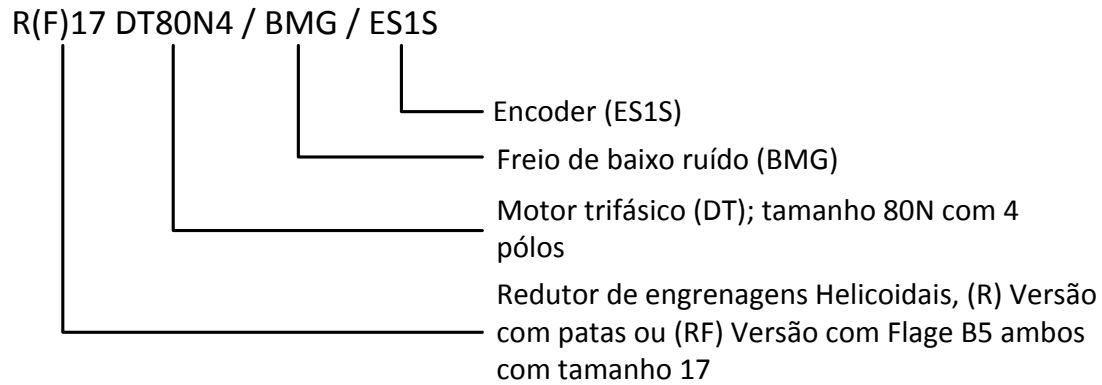



Figura 2.1: Designações e versões [SEW] para os moto-redutores dos eixos X e Y [11358858PT]

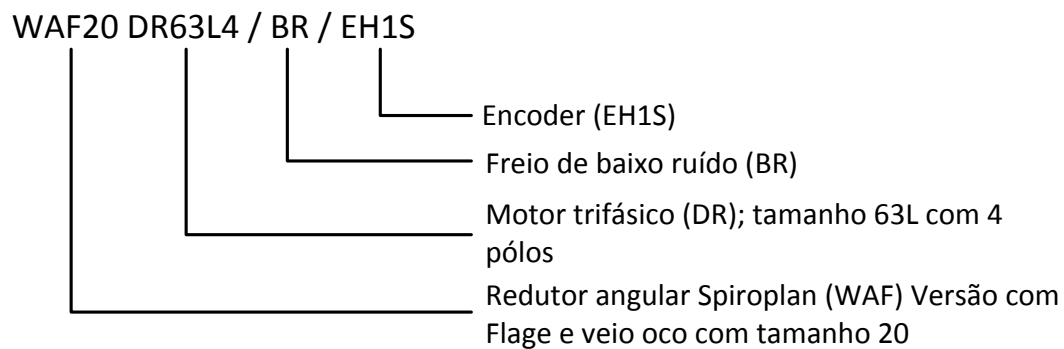



Figura 2.2: Designações e versões [SEW] para o moto-redutor para o eixo dos Z [11358858PT]

No que diz respeito aos significados das nomenclaturas escolhidas pela SEW-EURODRIVE para os tamanhos, tipos de flange e afins, estas apresentam uma lógica própria da empresa e pode ser consultado nos catálogos desta [SEW].

Os moto-redutores, que como o próprio nome indica são compostos por dois módulos, um motor trifásico e um redutor, em que a lado de saída do motor está ligado à entrada do redutor. Os motores usados são motores assíncronos ou de indução trifásicos com 4 pólos.

Os sistemas trifásicos de energia eléctrica são compostos por 3 tensões alternadas, nos quais a energia eléctrica é transmitida por meio da composição destes três sinais de tensão desfasados $\frac{2\pi}{3}$ radianos (120^0, ou seja 1/3 de cada ciclo).

Os motores de indução são basicamente compostos por um estator que é a parte estável e um rotor que é a parte móvel, entre eles encontra-se um espaço que normalmente se dá o nome de entreferro. O estator ou o circuito eléctrico estatórico é alimentado por um sistema de tensões trifásico que, provoca uma circulação de corrente nos condutores das bobines correspondentes a

cada fase, resultando num campo girante de força magnetomotriz que roda no espaço do entreferro à velocidade de sincronismo n_s:

$$n_s = \frac{120f}{p} \ (rot./\text{min}) \tag{2.1}$$

Em que f é a frequência da corrente alternada da rede eléctrica em Hz, que no caso da rede nacional é $50\ Hz$, finalmente p representa o numero de pólos

O movimento do campo magnético girante dá por sua vez origem à indução magnética, em que os condutores eléctricos das bobines de fase estatóricas (indutoras) induzem forças electromotrizes alternadas com uma amplitude e uma frequência que dependem da velocidade relativa entre o campo girante (n_s) e os condutores do rotor (n_r): $n_s - n_r$.

Estando o rotor curto-circuitado, as forças electromotrizes alternadas rotóricas dão origem a correntes eléctricas que circulam nos condutores do enrolamento rotórico, correntes estas que se encontram a circular no interior de um campo magnético, provocando assim o aparecimento de forças mecânicas sobre os condutores do rotor. Estas forças mecânicas combinam-se então para criar um binário (electromagnético) que faz rodar o rotor.

Por outro lado, o movimento do rotor tende a contrariar a causa que lhe deu origem, por isso, o rotor, por acção do binário electromagnético, tende a atingir a velocidade do campo girante mas sem nunca o atingir definitivamente devido ao atrito, $n_s > n_r$.

Assim, no motor de indução trifásico o rotor roda com uma velocidade ligeiramente inferior à velocidade do campo magnético girante, diferença esta que depende dos binários de carga (binário resistente) no veio do motor, e por isso é que se trata de um motor assíncrono, porque não existe uma relação constante entre a velocidade de rotação da máquina n_r e a frequência das grandezas eléctricas de alimentação f.

Ainda no que diz respeito ao campo magnético girante e a estas diferenças de velocidades, o rotor parece que desliza em relação ao campo magnético dando origem à grandeza deslizamento (s) - que é a razão entre a diferença de velocidade do rotor relativamente ao campo magnético e a velocidade do campo magnético:

$$s = \frac{n_s - n_r}{n_s} \tag{2.2}$$

Descrito, o módulo referente ao motor trifásico [Hughes, 2006], pode-se prosseguir com a drescrição do outro módulo dos moto-redutores usados, os redutores.

O redutor neste caso não é mais que um redutor de velocidade, ou seja, um dispositivo mecânico que reduz a velocidade de rotação do motor a que está acoplado aumentando deste modo o torque disponível. É composto basicamente por eixos de entrada e saída, rolamentos e engrenagens.

Os redutores da série R e RF são redutores de engrenagens helicoidais, isto é os dentes nas engrenagens helicoidais são cortados em ângulo com a face da engrenagem. E o redutor da série WAF é angular ou ortogonal, em que o eixo de saída é ortogonal ao eixo de entrada.

Acoplando os dois módulos optemos os nossos moto-redutores, representados na [Figura 2.3].

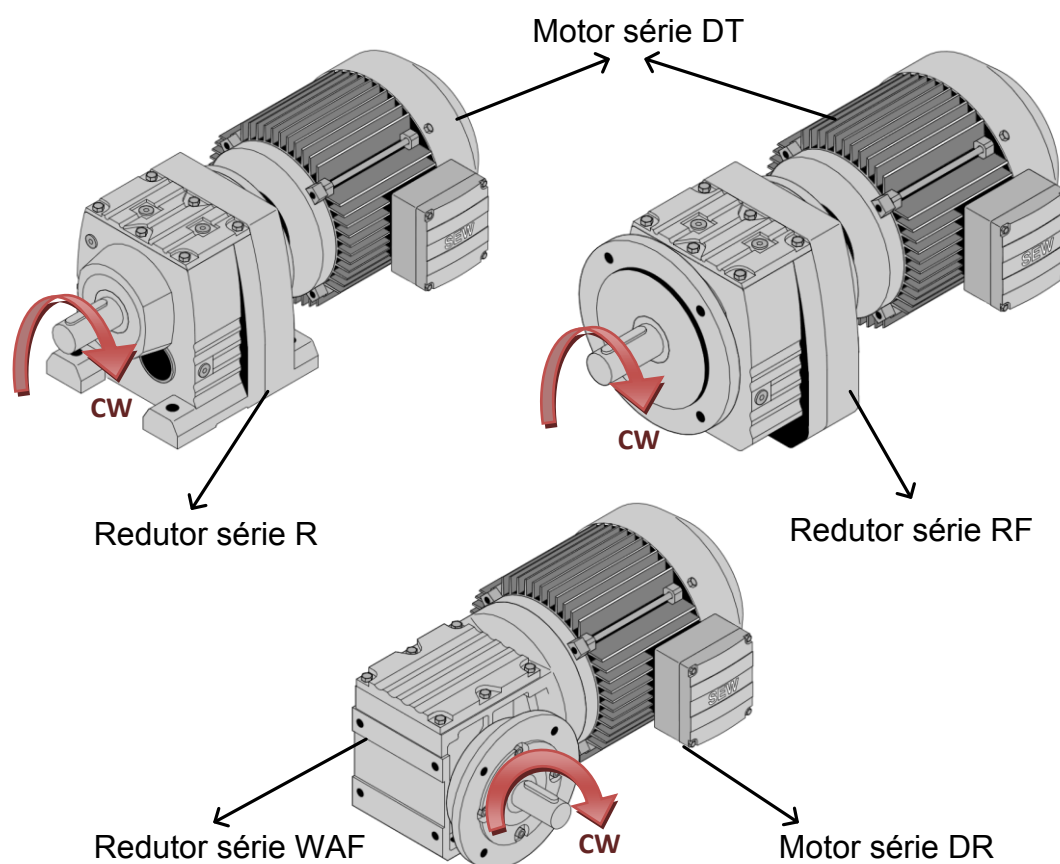



Figura 2.3: Moto-redutores da [SEW] usados, com indicação do sentido horário de rotação (CW) [EN16795210]

Olhando para o veio de saída de cada um dos moto-redutores, é importante ter noção do sentido de rotação para futuras montagens, assim temos que CW representa a rotação no sentido horário e CCW representa a rotação no sentido anti-horário. E também ter ainda em atenção a relação entre os valores referentes à velocidade de rotação à entrada do redutor (saída do motor n_r) e a velocidade à saída do redutor [Tabela 2.1].

Tabela 2.1: Relação entre a velocidade de rotação de entrada e saida dos redutores

Tipo de redutores	Velocidade de rotação à entrada do redutor (rpm)	Velocidade de rotação à saída do redutor (rpm)
Redutores R e RF	1380	196
Redutor WAF	1300	67

No Anexo A.1 encontra-se os desenhos explodidos do tipo de motores assim como o tipo de redutores que estamos a usar.

Estes três moto-redutores encontram-se equipados com encoders nos motores [Figura 2.4]. Assim a informação em relação ás posições que iremos obter depois, tem a ver com a posição relativo do motor. Para se obter a posição do moto-redutor é necessário levar em conta as relações apresentadas na Tabela 2.1.

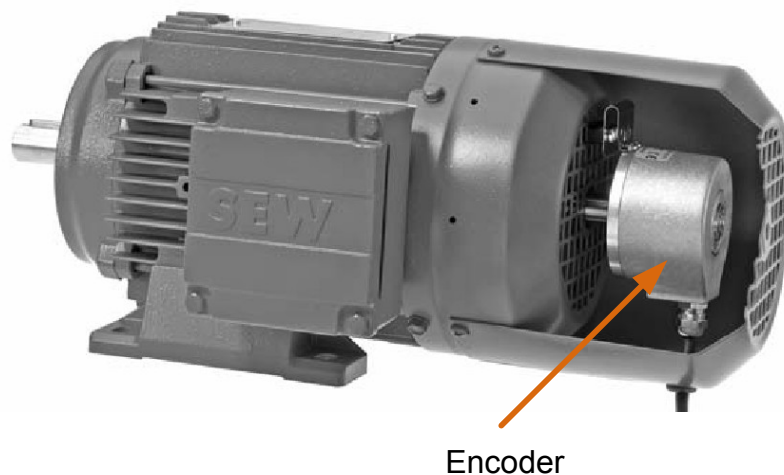



Figura 2.4: Localização do encoder no motor CA [EN16795210]

Os encoder do tipo ES e EH, são encoders incrementais rotacionais e têm 1024 incrementos por rotação, e possuem dois canais para contagem do número de incrementos e um canal de indexação [SEW] [EN16795210], contudo o controlador do conversor de frequência vai interpretar como sendo 4096 incrementos por rotação, logo o valor a reter é os 4096.

2.2 Conversores de frequência com controlo SEW-EURODRIVE

Os conversores de frequência, também conhecidos por inversores de frequência, são dispositivos electrónicos que recebem uma tensão da rede alternada sinusoidal, com uma frequência constante (neste caso $50Hz$), e transformam-na através de um rectificador de entrada em uma tensão contínua pulsada (onda completa), através de filtros passa depois para uma tensão contínua pura de valor aproximado, e finalmente é conectada aos terminais de saída pelos transístores (funcionando no modo corte ou saturação) transformando-a em uma tensão de amplitude e frequência variáveis sob a forma de onda sinusoidal.

Os conversores escolhidos para cada um dos três moto-redutores são os mesmos, e tem como referência a que está apresentada na [Figura 2.5].

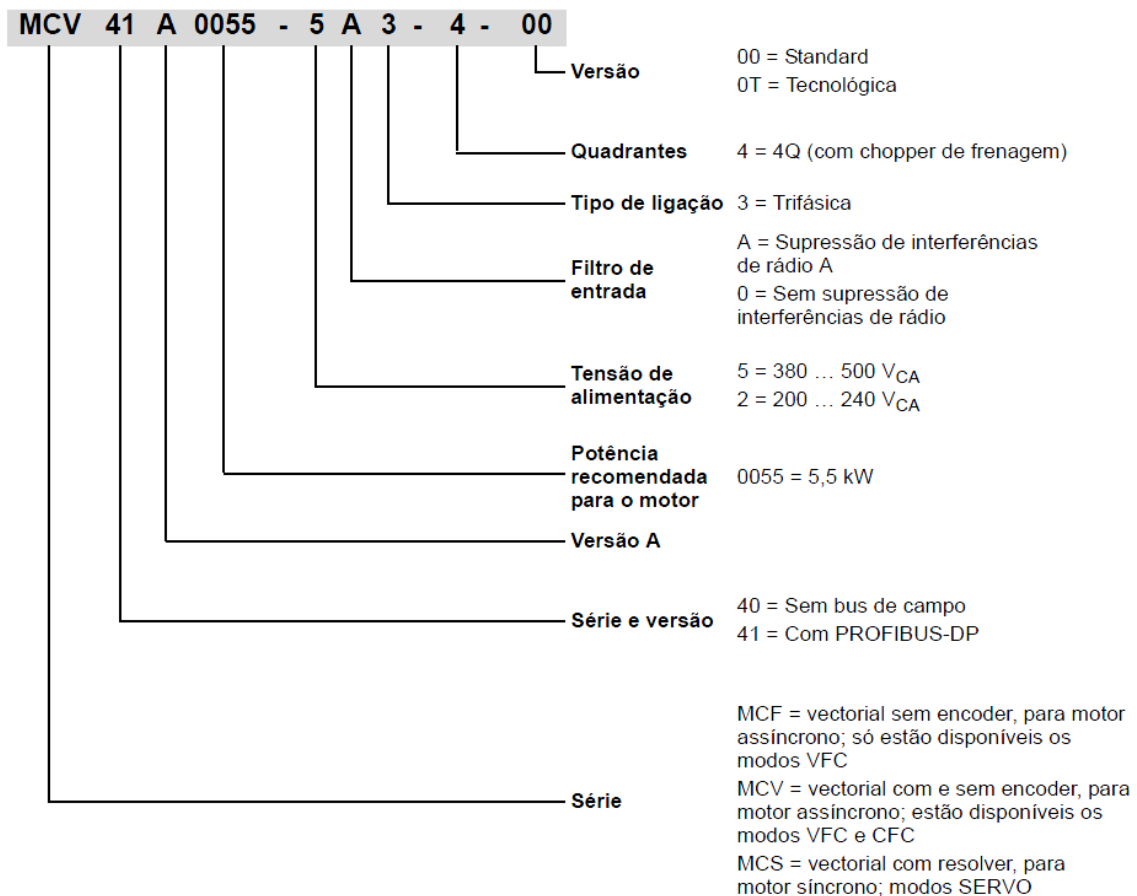



Figura 2.5: Interpretação da designação para o conversor de frequência da [11535040PT]

Estes conversores de frequência vêm com unidade central de processamento com memória integrada, o que permite programar modos de funcionamento, armazenar dados e parâmetros relativos de controlo. Dai a designação de conversores de frequência com controlo [11535040PT], e a imagem de um destes conversores pode ser vista na Figura 2.6.

Figura 2.6: Conversor de frequência com controlo [11535040PT]

Esta possibilidade de controlar e programar o conversor, é aquela que será explorada neste trabalho, pois ligando o PC ao conversor pelo adaptador de interface USS21A – RS232, descrito mais à frente no capítulo 3, pode-se controlar o conversor manualmente ou programar uma rotina que

ficará guardada na unidade de processamento/memória do conversor, que poderá mais tarde ser activada sem recorrer novamente ao PC.

Para comunicar com o conversor da [SEW], Usando o IPOS® [EN11320419] temos ao dispor um software, que é o Movitools, o qual irá ser descrito de seguida:

2.2.1 Software Movitools V.4.40

Este software, permite detectar o conversor a que estiver ligado, e a partir daí pode-se configurar este em função do motor que vai controlar, e ainda ter acesso em tempo real a diagnósticos sobre o motor ou mesmo o próprio controlador. Pode-se ainda, como foi dito, programar o controlador em linguagem C. Para tal, o programa possui um compilador de C onde se pode compilar e carregar o programa no controlador. A toda esta possibilidade de posicionamento e controlo sequencial através de programação a [SEW] deu o nome de IPOS® [EN11320419] [Figura 2.7].

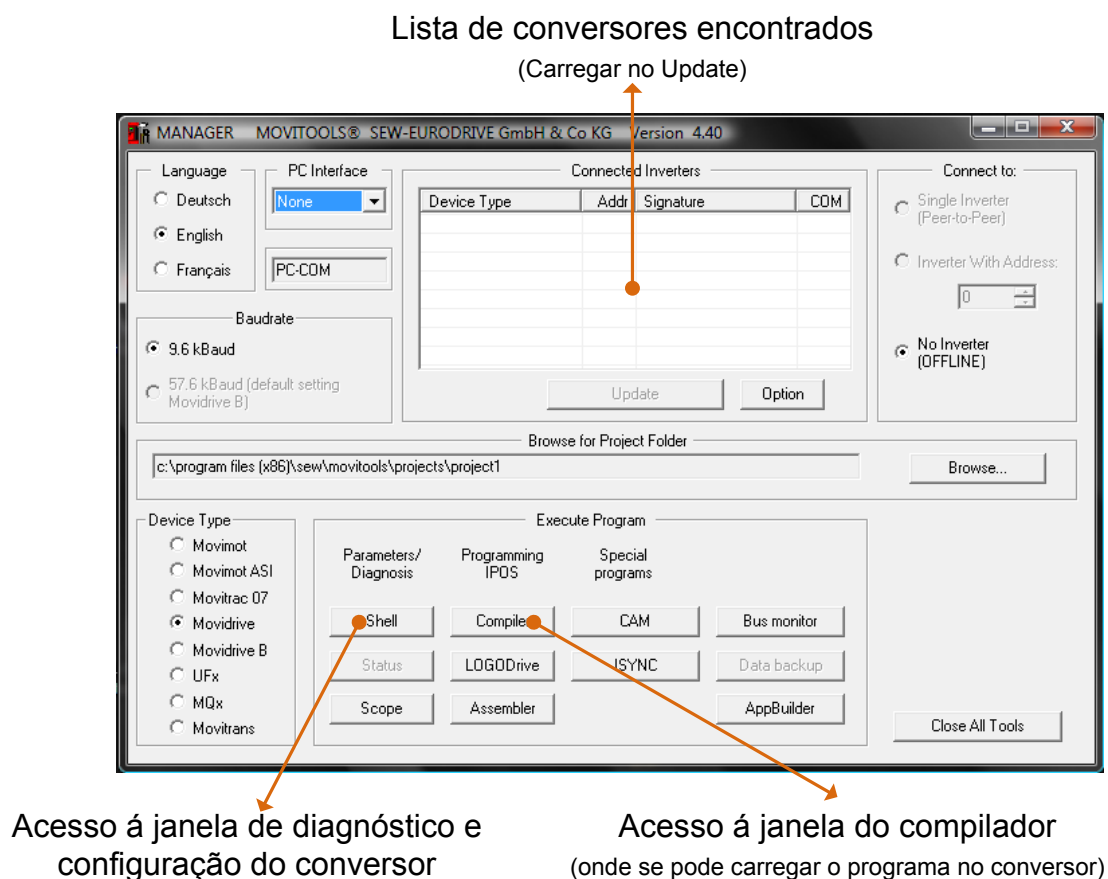



Figura 2.7: Interface do Programa Movitools V.4.40 [SEW]

Daqui para a frente, irá-se fazer referência apenas ás funcionalidades descritas acima. As restantes funcionalidades são simples e não criam dúvidas na altura da implementação. Para

informações mais detalhadas sobre todas as funcionalidades do programa consulte a documentação do próprio programa ou aceder a [SEW].

2.3 Controller Area Network (CAN)

Para fazer a ligação entre o PC e os conversores, optou-se por um sistema de interligação de dispositivos numa rede de acesso múltiplo a um meio partilhado denominado Controller Area Network (CAN). O CAN é um protocolo de comunicação em série síncrono, e um caso particular de fieldbus. O fieldbus neste contexto é visto como uma ligação múltipla de duas vias entre dispositivos de medida e/ou controlo. No entanto, e em geral, uma rede CAN serve para qualquer aplicação onde diversos dispositivos controlados por um micro controlador precisam de comunicar, de forma a completar uma tarefa comum – sistemas distribuídos [Venda, 2003]. As características essenciais do CAN, que conduziam à sua escolha são:

Difusões de mensagens transmitidas – Todos os nós vêem as mensagens transmitidas por qualquer nó que esteja na rede.

Acesso múltiplo ao bus (partilhado) – Todos os nós podem ler o bus ao mesmo tempo. Por outro lado na escrita no bus existe entre os nós um acordo mútuo, não destrutivo e sem atrasos de acesso de escrita (evitando assim colisões).

Definição de prioridades nas mensagens.

Múltipla detecção de erros (confinamento de falhas).

Retransmissão automática de mensagens corrompidas.

2.3.1 Arquitectura do sistema de comunicação

Partindo do modelo OSI (Open Systems Interconnection), as especificações da rede CAN definida pelas normas ISO 11898 e ISO 11519, descreve apenas a camada física e a camada de ligação de dados [Figura 2.8], ambas implementadas em hardware, deixando assim em aberto a possibilidade de cada grupo desenvolva o seu próprio padrão/software para a camada de aplicação.

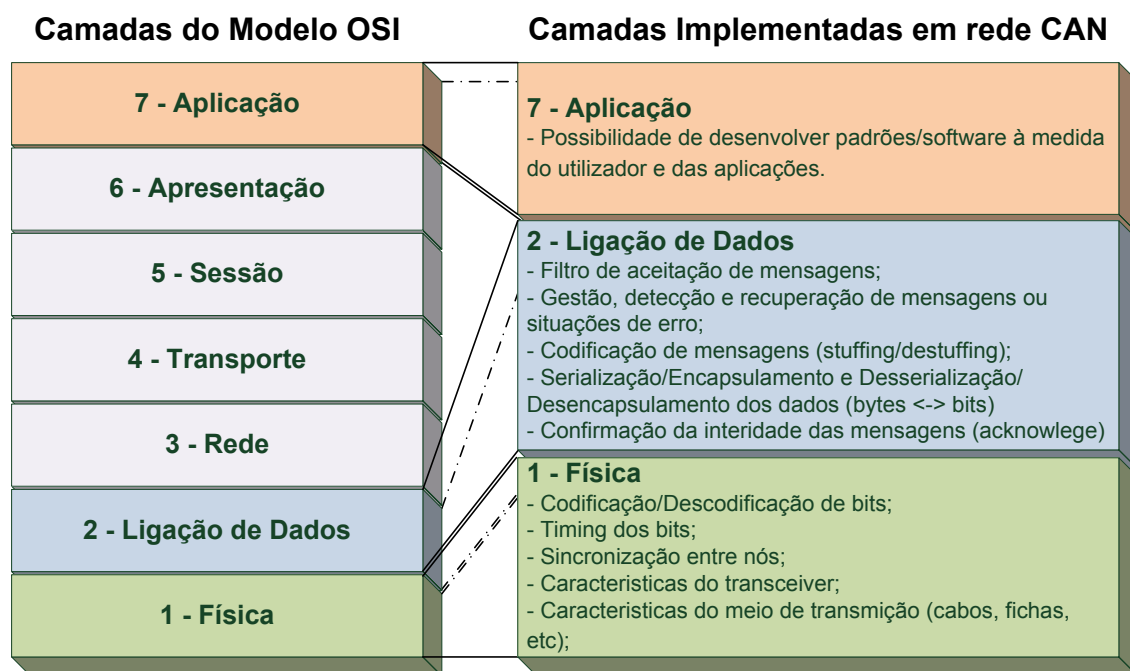



Figura 2.8: Modelo das camadas OSI vs CAN

Então do ponto de vista do hardware, a implementação de cada nó na rede CAN cobre como já foi referido, tanto a camada física como a camada de ligação de dados. Genericamente o hardware presente pode ser dividido em três blocos funcionais, o transceiver, o controlador CAN e o micro-controlador [Venda, 2003].

O "**transeiver"** que combina a transmissão e recepção, está normalmente implementado na camada física e no exterior do nó (tomada, fichas, cabos, etc) enquanto que o controlador encontra-se implementado na camada de ligação de dados. O micro-controlador tanto pode ser implementado na segunda camada como nas camadas superiores, e pode ser dedicado à interface de comunicação do próprio nó ou partilhado por outros periféricos.

2.3.1.1 Camada física

A camada física de um sistema de comunicação cobre os aspectos da transmissão física dos dados (bits) entre os nós da rede [Venda, 2003]. Olhando para a estrutura apresentada na [Figura 2.8] e seguindo essa ordem de raciocinio, pode-se desde já começar por falar da Codificação/Descodificação de bits.

Codificação/Descodificação de bits: Uma "stream" de bits num bus CAN é codificada em NRZ (Non Return to Zero), ou seja o bus pode estar num estado recessivo ou dominante, dependendo do bit que está a ser transmitido. Assim, o bus apresenta três níveis lógicos: o estado dominante, o estado recessivo e o estado *idle*, onde nenhum nó está a transmitir [CIA].

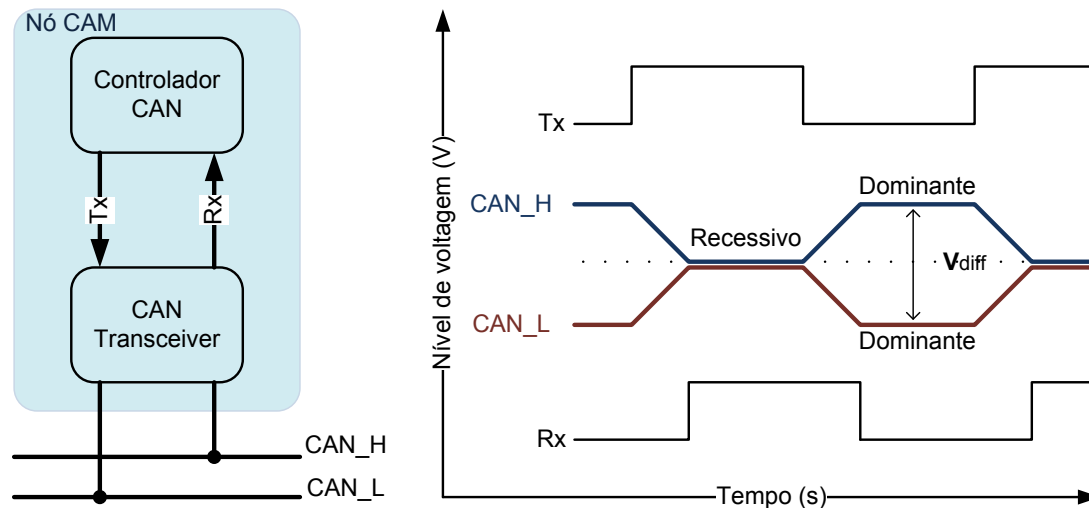



Figura 2.9: Representação em termos de voltagem os estados dominante e recessivo do CAN, e a contextualização física destes num nó.

Na Figura 2.9 podemos observar conforme a norma ISSO-11898 os dois estados lógicos de comunicação e a sua relação com os fios CAN_H e CAN_L [Richards, 2005].

Timing dos bits: Cada nó opera em sincronia com um oscilador de frequência pré-programada denominado time-quantum. Assim sendo, o tempo de transmissão de cada bit (bit-time) é definido como sendo múltiplo do time-quantum [Venda, 2003].

Sincronização entre nós: Se um nó estiver sincronizado com o bus, está por outro lado dessincronizado com o emissor devido aos tempos de propagação ao longo da rede. Por isso é necessário estender o bit-time em conformidade com a propagação, para garantir a coerência entre todos os nós da rede. Cada um dos nós espera pelos atrasos dos sinais e pela estabilização do bit antes de amostrar o bus e determinar que tipo de bit está no canal. O instante entre os dois segmentos de fase finais define o instante de amostragem do bus.

No caso de um nó detectar uma variação no nível do bus fora do segmento de sincronização, as durações dos segmentos de fase são reajustadas na altura, garantindo assim uma amostra mais fiável.

O CAN ainda possui um mecanismo denominado soft-syncroniztion que só ocorre uma vez em cada bit, e permite corrigir quer avanços de bits no tempo, quer atrasos de transição [Venda, 2003].

Em qualquer caso, a soma de todos os atrasos (processamento e propagação) do nó mais distante tem que ser menor do que o segmento de propagação. Este detalhe de sincronização vem limitar a relação ritmo de transmissão/comprimento do bus [Venda, 2003].

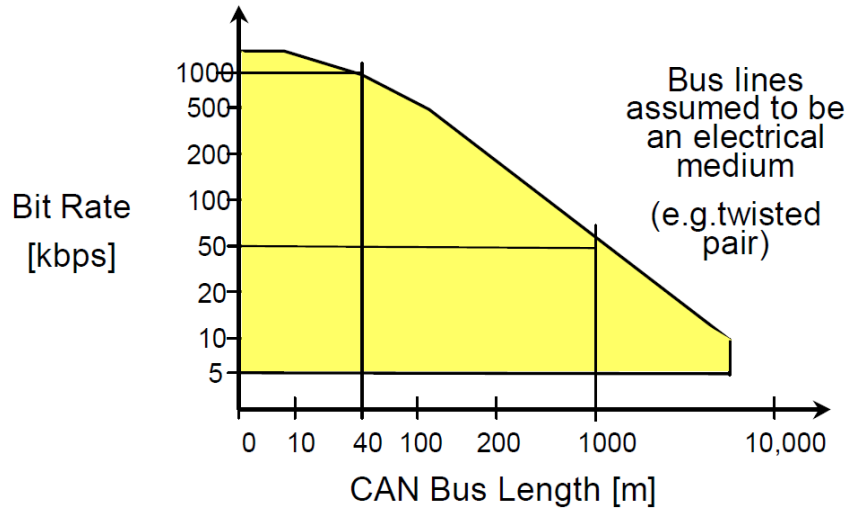



Figura 2.10: Relação entre a velocidade de transferência e o comprimento do bus[CIA].

Quanto maior for o bus, mais significativos são os atrasos de propagação e maior tem que ser o bit-time, reduzindo o ritmo de transmissão máximo [Figura 2.10].

Características do meio de transmissão: O standard ISO 11898-2 define o bus como uma linha bifilar simples (CAN_H e CAN_L – [Figura 2.9]) em que para minimizar as reflexões, é necessário terminar as extremidades com resistências [Figura 2.11]. Assim para o trabalho aqui exposto, comprimentos até 40 m num fio com resistência abaixo dos $60m\Omega/m$ pode-se usar resistências com um valor de 120Ω [Richards, 2005].

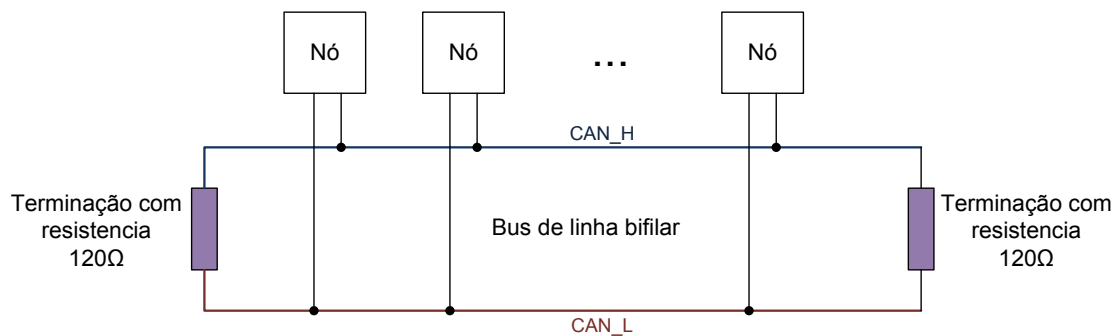



Figura 2.11: Topologia de uma rede CAN para bus com comprimento menor que 40m e resistência de fio normal.

Estas linhas como já foi dito apresentam três estados de actividade: *idle*, bit recessivo e bit dominante. Estes estados são definidos pelas tensões diferenciais introduzidas/medidas nas linhas ($V_{diff} = V_{CAN_H} - V_{CAN_L}$) pelos nós.

Esta natureza diferencial da medida minimiza interferências electromagnéticas, dado que os dois fios do bus estarão fisicamente perto um do outro e, na presença de interferência, os efeitos sentidos em ambos serão semelhantes, e como a tensão é medida diferencialmente, o valor dessa interferência desaparece [Venda, 2003].[Venda, 2003]

2.3.1.2 Camada de ligação de dados

É na camada de ligação de dados que se impõe formatos às mensagens que circulam pelo bus e ainda detecta e, por vezes também corrige, erros que possam surgir na camada física.

No que diz respeito as mensagens, existe actualmente duas versões de mensagens, as *standard* e as *extended*, que serão explicadas mais à frente. Por isso o hardware pode ser implementado tendo em conta estas duas versões. Foram diferenciadas três especificações de protocolo de ligação representadas na Tabela 2.2:

Tabela 2.2: Relação entre os protocolos de comunicação e as duas versões de mensagens

Tipo de Protocolo de Ligação	Mensagens *Standard*	Mensagens *Extended*
2.0 A	Recebe e Transmite	Reconhece como erro
2.0 B Passivo	Recebe e Transmite	Recebe
2.0 B Activo	Recebe e Transmite	Recebe e Transmite

Filtro de aceitação de mensagens: Nesta camada existe uma subcamada que filtra as mensagens que recebe, isto é, decide consoante um identificador se a mensagem é entregue à camada acima ou não. Todo este mecanismo pode ser programado pelo utilizador nas camadas superiores.

Serialização/encapsulamento, desserialização/desencapsulamento: As informações das camadas superiores têm de ser serializadas e encapsuladas antes de irem para o bus, isto é, é preciso transformar cada byte recebido numa sequência de bits, para possibilitar a transmissão em série no bus e vice-versa.

Stuffing/destuffing: Sempre que o transmissor detectar a presença de 5 bits consecutivos da mesma polaridade numa stream de bits (após a serialização), introduz um bit de polaridade inversa que vai ser automaticamente removido pelos receptores. Isto acontece porque no caso de uma sequência de bits iguais (recessivos ou dominantes), os nós podem ter dificuldades na sincronização entre si. Esta regra de stuffing pode ser utilizada para uma verificação de integridade [Venda, 2003].

Carrier Sense Multiple Access/Deterministic Collision Resolution (CSMA/DCR) é o que evita as colisões no acesso múltiplo, isto é, para evitar que dois ou mais nós enviem ao mesmo tempo uma mensagem diferente, o CSMA/DCR determina prioridades. Para tal, cada mensagem leva consigo um identificador, uma identidade que é atribuída pelo nó. Durante a transmissão do identificador da mensagem, o algoritmo CSMA/DCR decide de forma unívoca, não destrutiva e sem atrasos ou retransmissões quem transmite ou não a mensagem.

Data-Frame: É o nome que se dá a tudo o que temos vindo a referenciar como mensagens, as tramas de dados (*data frames*) são as unidades de comunicação entre as camadas de ligação de

dados dos vários nós [Figura 2.12]. Sempre que um nó pretende enviar dados para o bus, fá-lo através do encapsulamento da informação em uma ou mais tramas, em que cada uma pode transportar 1 a 8 bytes de informação.

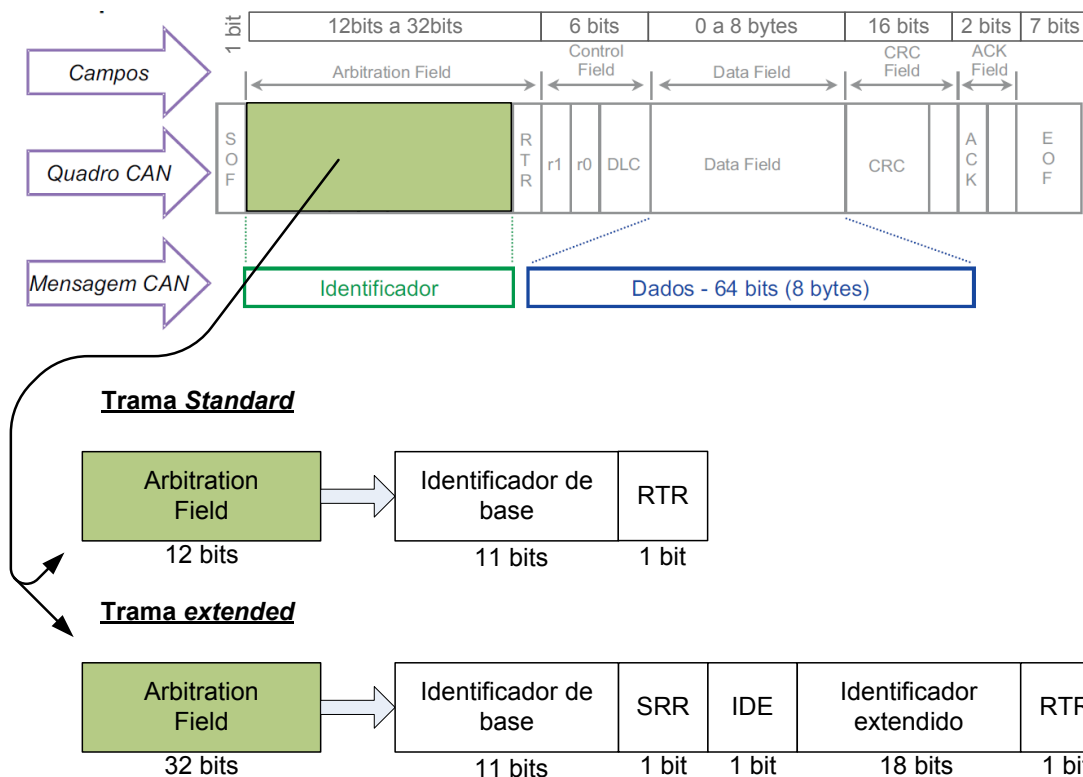



Figura 2.12: Representação de uma trama de dados e as diferenças entre Standard e Extended

Entre duas mensagens sequenciais existe um conjunto de três bits recessivos a que se chama *Intermission Frame Space,* que não se encontra representada na Figura 2.12.

Então, cada *data-frame* começa com um bit *Start of Frame* (SoF), que foi em tempos utilizado para uma sincronização forçada, contudo é desnecessário pois actualmente o algoritmo CSMA/DCR usa o campo de identificação (*arbitration field*) para determinar o nó a transmitir a mensagem. E no campo de identificação que se encontra a diferença entre trama de dados *standard* e *extended* [Figura 2.12], que é identificado pelo bit IDE (*Identifier Extension*). O RTR (*Remote Transmission Request*) é utilizado no estado recessivo para marcar as mensagens como RTRs. O *Substitute Remote Request* (SRR) serve apenas para substituir o RTR que apareceria se a trama fosse *standard*.

O campo de controlo (*control field*) especifica o comprimento em bytes da mensagem. Mais à frente o campo de dados (*data field*), que é onde vai a informação que se quer enviar. Depois segue-se o campo de CRC (*cyclic redundancy check*) que é onde é guardada a informação referente à detecção de erros em determinadas zonas da trama. Segue-se o campo de confirmação ACK (*acknowledge field*) que como o próprio nome indica é utilizado para dar a conhecer ao transmissor

que a mensagem foi recebida pelo menos por um nó sem erros de bus. Finalmente, a mensagem é terminada pelo *End of Frame* (EoF)[Venda, 2003].

Tratamento de erros: Cabe aos dispositivos físicos e às camadas de hardware a introdução de mecanismos de tolerância a falhas na comunicação. Num dado nó k o tratamento de erros processa-se da seguinte forma [Venda, 2003]:

1. O nó k detecta o erro local;

2. É enviada uma "error flag" para globalizar o erro – 6 bits da mesma polaridade para forçar um erro de stuffing nos outros nós;

3. Depois da "error flag" envia-se uma "overlapping error flag", seguida de um "error delimiter";

4. A mensagem é descartada em todos os nós;

5. Os contadores de erro são incrementados em todos os nós;

6. A transmissão é repetida automaticamente;

Os erros são portanto assinalados e globalizados assim que são detectados. Através deste processo de tratamento de erros, a coerência dos dados é garantida, caso nenhum nó esteja em estado passivo ou em bus off, à parte de um efeito pernicioso no bit final da mensagem.

De seguida iremos falar do dispositivo usado para fazer a ligação entre o PC e o CAN, este dispositivo pode-se enquadrar na **Camada de Ligação** do modelo das camadas do OSI.

2.4 Dongle CAN/USB – CANBUS da LAWICEL AB

O CANUSB é um dongle da LAWICEL que faz a ponte entre o PC e o CAN-bus [Figura 2.13]. O driver instalado foi o *Direct Driver* (D2XX), que usa um DLL (biblioteca de ligação dinâmica) para comunicar com o CANUSB.

Figura 2.13: Dongle CANUSB da LAWICEL – faz a ponte entre o PC e o CAN-bus

O CANUSB tem como protocolo de ligação 2.0B activo, logo é compatível com tramas standard assim como extended. No que diz respeito à estrutura de dados, o CANUSB guarda as mensagens num tipo de fila FIFO (First In, First Out) e reporta ainda erros de comunicação. Não necessita de alimentação externa, pois é alimentado pela porta USB.

2.4.1 Limitações

Este produto garante bons desempenhos para taxas de transferências menores ou iguais a $250kbps$, contudo pode-se usar até $1Mbps$. O CANUSB como foi dito guarda as mensagens num tipo de fila FIFO, e guarda quer as que envia (no caso de o bus do CAN estar ocupado) assim como aquelas que recebe (esperar que sejam lidas pelo PC). No envio de tramas o FIFO guarda até 8 tramas (*standard* ou *extended*), enquanto que o receber, guarda até 32 tramas (*standard* ou *extended*)

2.4.2 Funções usadas do canusbdrv.dll.

Na pasta de sistema deve se encontrar o ficheiro canusbdrv.dll (driver de dispositivo) após a instalação. Depois de importar a driver para o código no qual se está a desenvolver o software que irá comunicar com o CANUSB, podemos então comunicar com o CAN-bus usando as funções contidas nessa mesma driver usando assim o CANUSB como elo de ligação. Assim as funções usadas neste trabalho foram [CANUSB]:

2.4.2.1 canusb_Open ()

Esta função, abre um canal com a camada física do CAN. É possivel com esta função abrir mais canais de comunicação com a mesma camada física, basta para tal usar esta função várias vezes sem a fechar. De cada vez é criado um canal virtual, no entanto é conveniente fechar o mesmo numero de vezes o canal (a ordem não é importante)

A função em linguagem C++.NET é definida da seguite forma:

```
Int32 canusb_Open( char * szID, char * szBitrate, UInt32 acceptance_code,
UInt32 acceptance_mask,UInt32 flags );
```

Para abrir canais virtuais, a interface tem de ser indicada no *szID*, em que este é o número de série do adaptador, ou então preencher com zero, para a primeira utilização.

A função retorna um valor que chamamos de "*drvhandle*" – valor para identificar o canal aberto.

O *szBitrate* é o baudrate, ou taxa de transferência, em que "250" significa 250 $kpbs$.

O *acceptance_code* e o *acceptance_mask* funcionam como filtros de mensagens, e usou-se os seguintes valores para aceitar todas as mensagens:

```
const UInt32 CANUSB_ACCEPTANCE_CODE_ALL   =0x00000000;
const UInt32 CANUSB_ACCEPTANCE_MASK_ALL   =0xFFFFFFFF;
```

No que diz respeito às flags usou-se aquela que faz com que o driver escolha a melhor solução:

```
const UInt32 CANUSB_FLAG_TIMESTAMP=0x0001;
```

2.4.2.2 canusb_Close ()

Esta é a função responsável pelo fechar do canal aberto pela função *canusb_Open ()*, descrita em cima. E a função em linguagem C++.NET é definida como:

```
Int32 canusb_Close(int drvhandle)
```

Se houver alguma mensagem no FIFO ao fechar o canal esta perde-se. A função retorna ≤ 0 se falhar e > 0 no caso de sucesso.

2.4.2.3 canusb_Read ()

Esta função permite ler do canal identificado com *drvhandle* as mensagens enviadas pelos outros nós na rede CAN-bus. E a função em linguagem C++.NET é definida como:

```
Int32 canusb_Read(int drvhandle, CANMsg *msg)
```

A função retorna ≤ 0 se falhar e > 0 ou *ERROR_CANUSB_OK* no caso de sucesso. Se por outro lado retornar *ERROR_CANUSB_NO_MESSAGE* é porque não existe nenhuma mensagem para ser lida no FIFO. Os valores para estes códigos de erros são definidos como:

```
static const Int32 ERROR_CANUSB_OK = 1;
static const Int32 ERROR_CANUSB_NO_MESSAGE = -7;
```

Em que a estrutura para a mensagem é definida como:

```
typedef struct
{
      UInt32 id;          // ID da mensagem
      UInt32 timestamp; // timestamp in milliseconds
      char   flags;      // Trama standard= 0.
      char   len;        // tamanho em Bytes da trama
      char   data[ 8 ]; // Mensagem propriamente dita
} CANMsg;
```

2.4.2.4 canusb_Write ()

Esta função permite escrever através do canal identificado com *drvhandle* as mensagens que se prentende enviar para o CAN-bus. E a função em linguagem C++.NET é definida como:

```
Int32 canusb_Write(int drvhandle, CANMsg *msg )
```

Para se aceder à mensagem, usa-se a mesma definição de estrutura que *o canusb_Read (),* mas com nome diferente na altura da implementação No que diz respeito aos valores retornados pela função, os códigos de erro são praticamente idênticos, exceptuando que esta função tem mais um código de erro o *ERROR_CANUSB_TX_FIFO_FULL* para quando é impossível escrever porque o FIFO está cheio, e é definido:

```
static const Int32 ERROR_CANUSB_TX_FIFO_FULL = -5;
```

2.4.2.5 canusb_Flush ()

Esta função permite limpar o buffer de saída, ou seja permite limpar o FIFO de saída. A função em linguagem C++.NET é definida como:

```
Int32 canusb_Flush( CANHANDLE h, Char flushflags)
```

A função retorna ≤ 0 se falhar e > 0 no caso de sucesso. No que diz respeito a flags usou-se somente a flag *FLUSH_DONTWAIT* que limpa o buffer sem esperar que mensagens que estejam em espera sejam enviadas, e é definida como:

```
static const Char FLUSH_WAIT = 0x00;
```

2.4.2.6 Estrutura das mensagem

Como se viu no subcapítulo 2.4.2.3, a estrutura da mensagem a enviar para o CAN-bus é composta por um ID, dados e flags (timestamp, flags e len). Assim o ID corresponde na [Figura 2.12] ao identificador, e os dados à trama de dados de 8 bytes, as flags servem para o CANUSB interpretar se a trama é estendida ou não, o número de bytes que se envia e a estampa temporal.

O que importante aqui referir é que os dados estão partidos em duas sequências de 4 bytes em que a primeira foi escolhida para transportar os códigos, que serve para identificar o que vai nos 4 bytes seguintes, ou então para dar uma ordem específica da qual os seguintes 4 bytes não terão importância. Para consultar estes códigos e variáveis envolvidas ir para [Anexo A4]

A título de exemplo, para se enviar uma posição para um dos servos, é necessário que o primeiro grupo de 4 bytes contenha o código que indique que o valor a seguir é uma posição. Se por outro lado se quiser enviar uma ordem para que os motores se mexam, envia-se o código nos primeiros 4 bytes e os restantes não serão lidos.

2.5 Ficheiros CAM

O software protótipo desenvolvido [capítulo 4], lê ficheiros com coordenadas que descrevem a trajectória que a fresa deve tomar de forma a esculpir o material em bruto. Isto significa, que são instruções para um movimento e não uma nuvem de pontos que descrevem a geometria do produto final.

Os ficheiros para serem aceites devem conter três coordenadas por linha, uma para o eixo dos *xx*, outra para o eixo dos *yy* e mais uma para o eixo dos *zz.* A ordem pela qual se encontram ao longo da linha não importa, devem é estar separadas entre si por um espaço. As coordenadas podem ser negativas ou positivas, mas todas devem ser precedidas por uma letra que represente o eixo a que pertence. Por exemplo a cordenada $(30, -40.5, 87)$ apareceria como:

X30 –Y-40.5 Z87

A trajectória entre estes pontos será vista como uma interpolação linear, ou seja as ligações entre os pontos serão segmentos de recta. Assim sendo, o que se pretende então, é que a fresa se mova com uma velocidade constante entre estes pontos, e que se mantenha sempre na trajectória delimitada pelo segmento de recta.

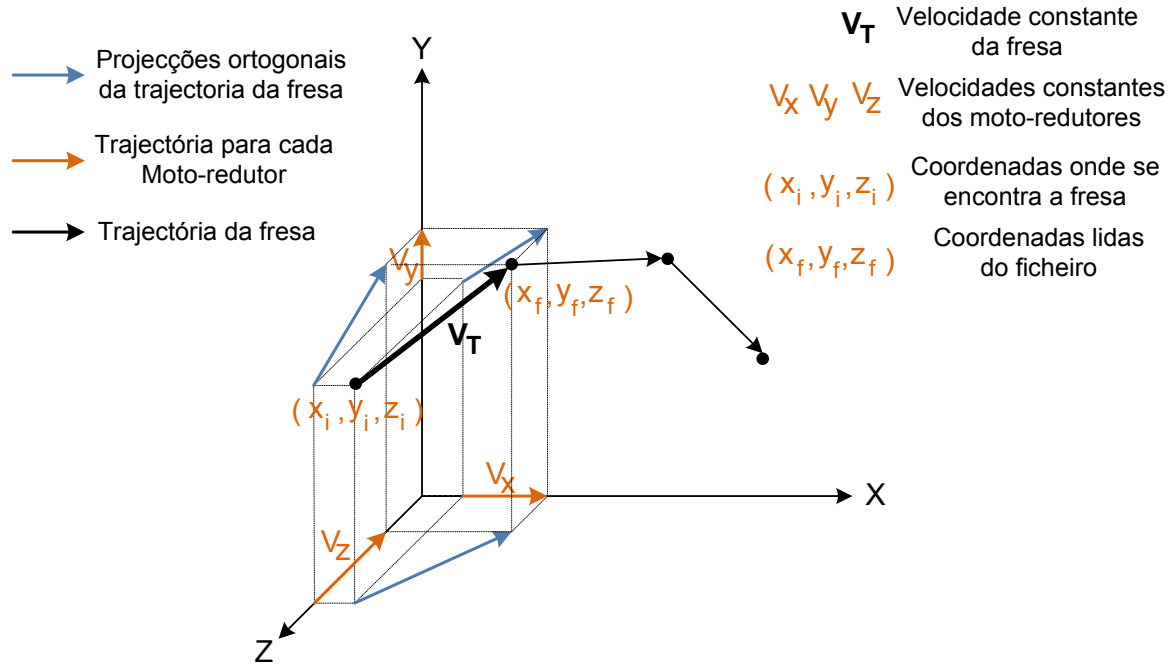



Figura 2.14: Descrisão do movimento da fresa entre pontos por interpolação linear

Com base na Figura 2.14, vê-se que o que vai determinar que a fresa cumpra as condições impostas pela interpolação linear, são as velocidades dos moto-redutores, que terão de ser tais que percorram as suas distâncias no mesmo espaço de tempo. O que significa que para cada eixo as velocidades a aplicar a cada motor são:

$$\begin{cases} V_x = V_T \dfrac{x_f - x_i}{\Delta r} \\ V_y = V_T \dfrac{y_f - y_i}{\Delta r} \\ V_z = V_T \dfrac{z_f - z_i}{\Delta r} \end{cases} \tag{2.3}$$

Em que Δr, representa a distância percorrida pela fresa é pode ser calculada usando:

$$\Delta r = \sqrt{(x_f - x_i)^2 + (y_f - y_i)^2 + \left(\frac{U_R}{U_{WAF}}(z_f - z_i)\right)^2} \tag{2.4}$$

Em que, U_R e U_{WAF} são o numero de incrementos que cada encoder conta para que os redutores da séries R ou RF e da série WAF [Figura 2.3] dêem uma volta completa. Ou seja são as unidades de cada eixo.

Capítulo 3

3 Desenvolvimento experimental

Neste capítulo iremos primeiro apresentar a concepção geral referente à montagem efectuada no laboratório de todo os dispositivos aqui descritos, e segundo a qual se efectuou todos os testes. De seguida, apresenta-se os esquemas de ligação entre cada dispositivo representado na concepção detalhadamente. Segue-se uma referencia a como carregar os códigos desenvolvidos neste trabalho nos conversores de frequência, usando o software MOVITOOLS, assim como que ligação é necessária para tal. E finalmente, apresenta-se montagem da bancada de ensaios.

3.1 Concepção geral

Para se ter uma ideia geral de como é que todos os dispositivos e modos de comunicação estiveram dispostos durante o desenvolvimento do software de controlo DACS-OPMG V1.0 (descrito no capítulo 4), e respectivos testes no laboratório, apresenta-se de seguida o seguinte esquema [Figura 3.1]:

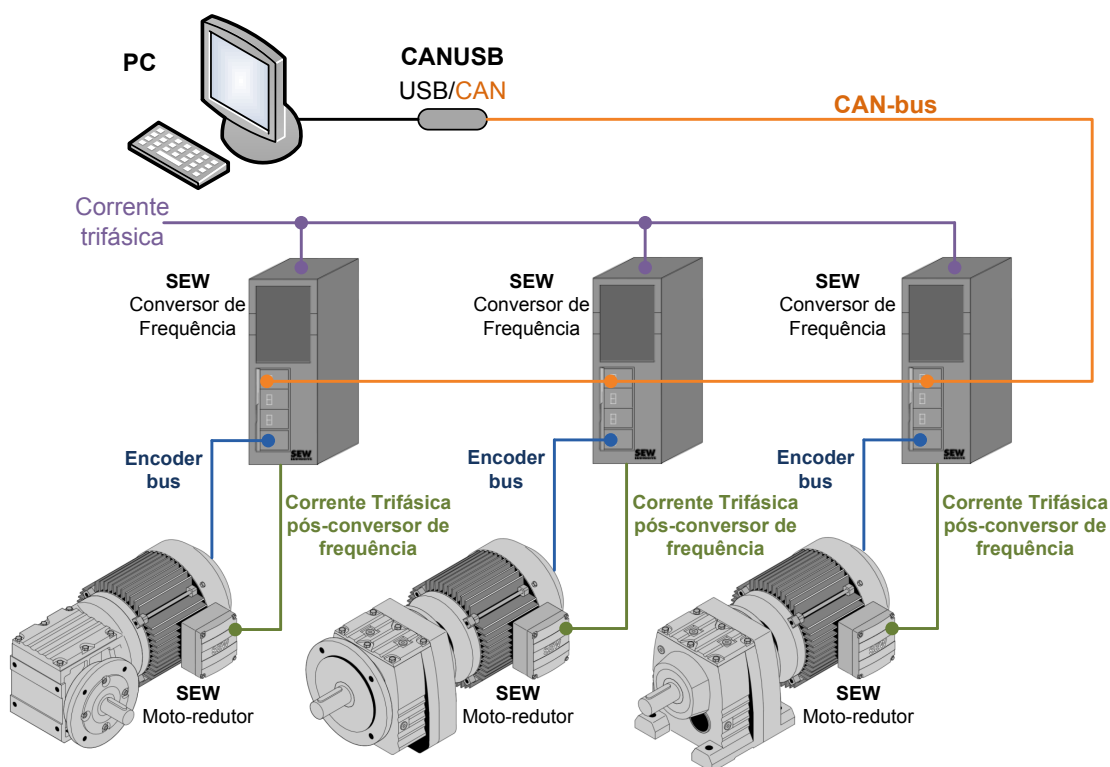



Figura 3.1: Esquema geral de ligações

Em seguida e durante este capítulo irá se descrever as ligações apresentadas na [Figura 3.1] fazendo referência pela cor.

A linha cor de laranja representa o CAN-bus, que possibilita a comunicação entre 4 nós, 3 deles são os conversores da SEW e o 4 o conjunto CANUSB e PC (ligação feita pela porta USB do PC). É por este bus que se comunica com os conversores de frequência usando o software DACS-OPMG, e por onde também se recebe o feedback dos conversores. A linha com a cor violeta, representa a alimentação dos conversores à rede de sistema trifásico de energia eléctrica. A linha verde, por sua vez, representa a alimentação dos moto-redutores com uma corrente trifásica pós-conversor, isto significa que a corrente inicial da rede foi transformada em uma tensão de amplitude e frequência variável conforme as necessidades de velocidade de rotação que se espera tirar do motor. Por fim, a linha azul representa a ligação de comunicação entre o encoder e o conversor, por onde como já se falou, é comunicada a posição relativa do motor.

3.2 Esquema das ligações eléctricas

Figura 3.2: Esquema das ligações eléctricas entre Conversor de frequência e o motor.

Nesta secção apresenta-se então um esquema representativo das ligações que na Figura 3.1 estão representados com a cor violeta e verde, ou seja, os esquemas das ligações eléctricas. Na Figura 3.2 encontra-se assim, segundo o fabricante, o referido esquema representativo das ligações eléctricas entre os conversores de frequência e os motores em pormenor. Assim, para ligar os três motores aos conversores de frequência basta seguir o esquema para cada um deles, pois as referências usadas são válidas para os três conjuntos.

A relé é introduzida como um interruptor, com objectivo de servir de travão para o motor, e que como características deve ter uma tensão de controlo de $24V\ CC$ para uma tensão de comutação de $380V\ CA$.

Para ver a estrutura do conversor de frequência, e onde se encontram fisicamente cada uma das ligações que estão referidas na [Figura 3.2] consultar o [Anexo A.2].

3.3 Esquema das ligações Encoder-Conversor

Segue-se a representação da ligação entre os encoders e os conversores de frequência [Figura 3.3], em que olhando para a [Figura 3.1], representa as ligações a azul.

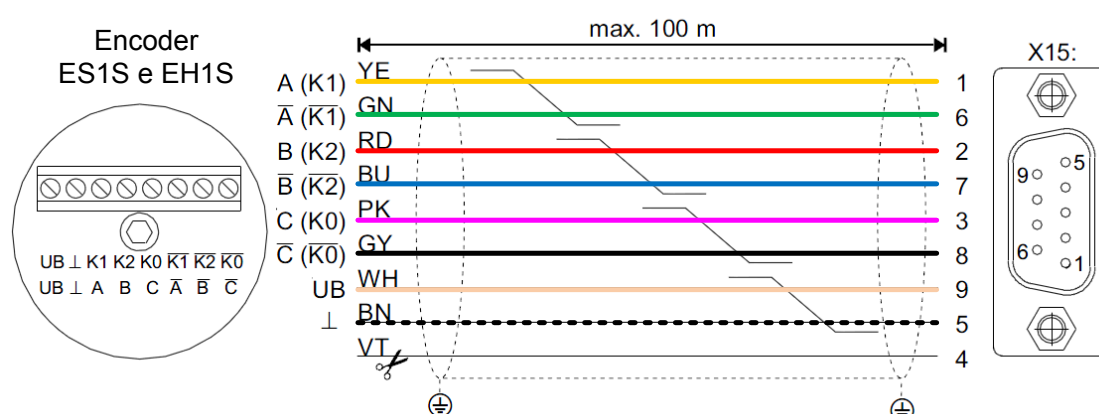



Figura 3.3: Esquema para a ligação entre os encoders ES1S e EH1S, e os conversores de frequência

Para a localização do enconder no motor ver Figura 2.4. No que diz respeito á localização do X15 no Conversor de frequência consultar Anexo A.2.

3.4 Esquema das ligações X10 do Conversor e as ligações CAN-bus

Mais uma vez recorrendo à Figura 3.1, esta parte vai tratar das ligações relacionadas com aquelas que têm a cor de laranja, e alem dessas ligações iremos também apresentar no esquema as

ligações referentes à régua de terminais electrónicos X10 do conversor de frequência da SEW, e mais uma vez, para verificar a localização do X10, consultar Anexo A.2.

Assim sendo, na [Figura 3.4] encontra-se representado as ligações que se tem de efectuar para se obter um CAN-bus ligando todos os conversores de frequência, sem esquecer terminar as pontas para evitar reflexões no bus. Os conversores de frequência recebem estas ligações nos X10, e é aqui que se também implementa os interruptores para desligar e ligar os motores manualmente, assim como, as interrupções detectadas pelo IPOS® no controlador dos conversores de frequência. Estas últimas interrupções permitem interromper a rotina do código que estiver a correr no controlador, podendo assim implementar uma nova rotina, no caso em concreto deste trabalho, usou-se esta interrupção para poder efectuar calibrações (explicado no capítulo 4).

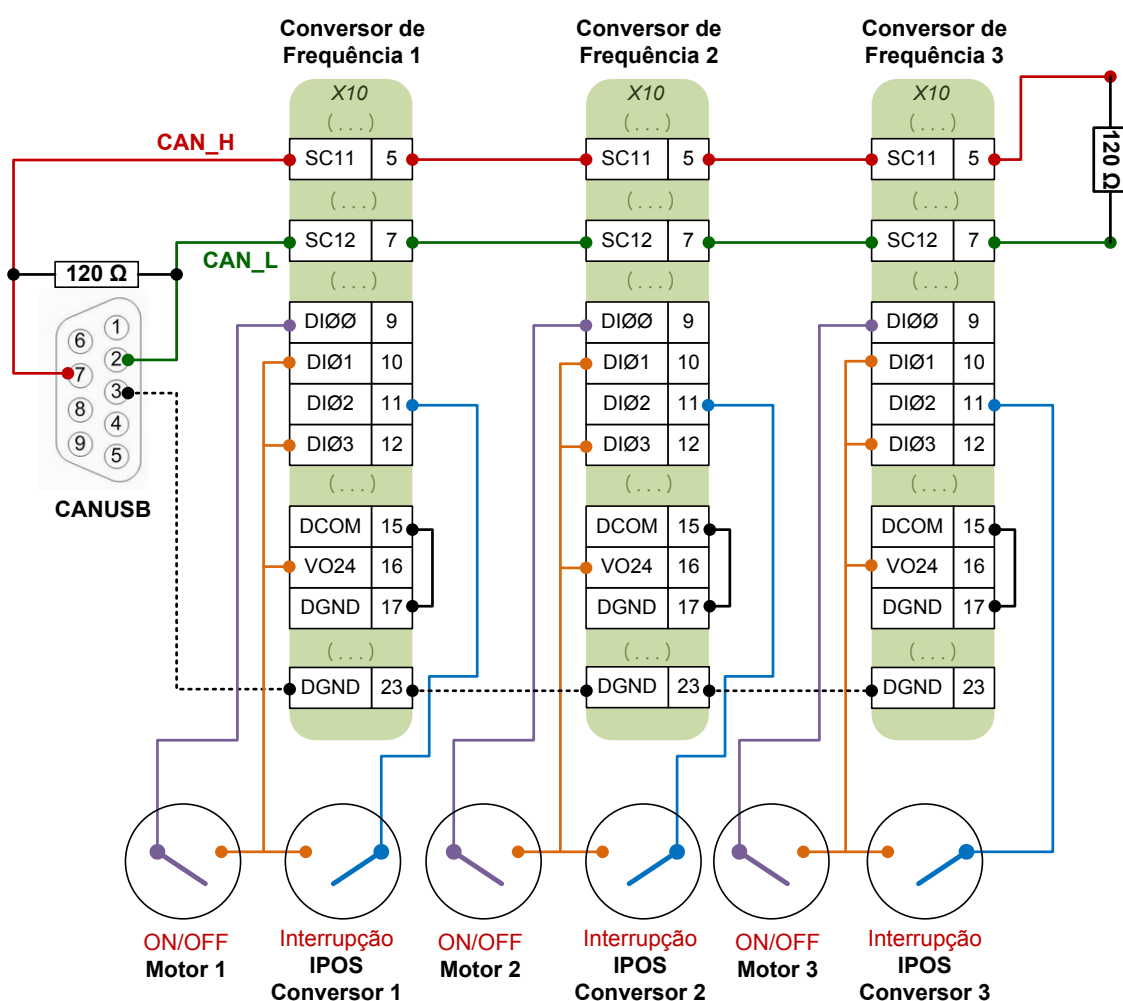



Figura 3.4: Esquema das ligações do X10 dos conversores e as ligações CAN-bus ao longo dos conversores

3.5 Carregamento do código nos controladores dos conversores de frequência

Para carregarmos o código desenvolvido para os controladores dos conversores, primeiro temos de efectuar uma ligação entre o PC e os controladores, como está apresentado na Figura 3.5:

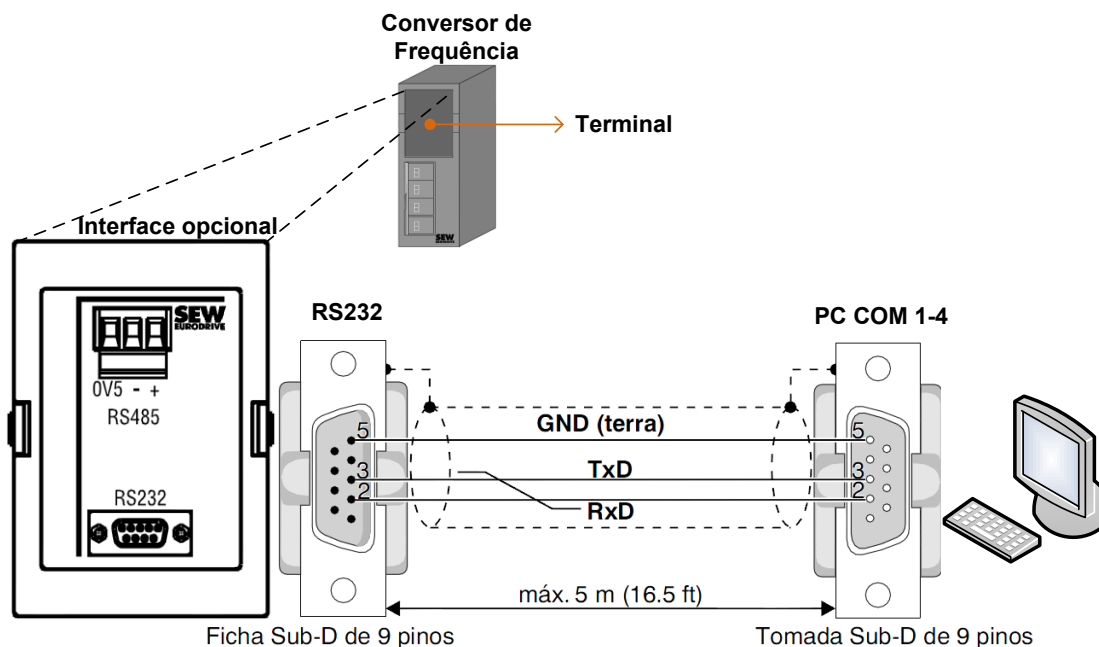



Figura 3.5: Esquema para ligar o PC ao controlador do conversor de frequência

Para fazer esta ligação é necessário ligar a interface opcional no Terminal do Conversor de frequência (ver Anexo A.2 para localização exacta). Depois é Ligar um cabo de uma das portas COM 1-4 do PC ao RS232.

Depois de feita esta ligação é possível obter informações do controlador do conversor, tais como, tipo de conversor, posição relativa do motor, velocidade do motor, etc. E claro, passa a ser possível carregar os códigos no controlador do conversor, para tal é necessário usar o programa MOVITOOLS V4.40. Temos duas maneiras de carregar o código, ou carregamos um código directamente do compilador do programa, ou através da janela de diagnostico e configuração do conversor denominada *Shell*.

Olhando para a Figura 2.7, podemos ver como aceder à janela que contem o compilador [Figura 3.6], onde com o código já desenvolvido, é possível verificar se existe erros de compilação ou fazer alterações de ultima hora neste. Depois de tudo verificado, Pode-se então carregar o programa no controlador, contudo este simples acto não deixa o programa activo, para isso é necessário dar ordem de iniciação do programa, uma vez feito isto, o programa fica a correr no controlador sempre que este tiver ligado à corrente, não sendo mais necessário estar ligado ao PC [Figura 3.6].

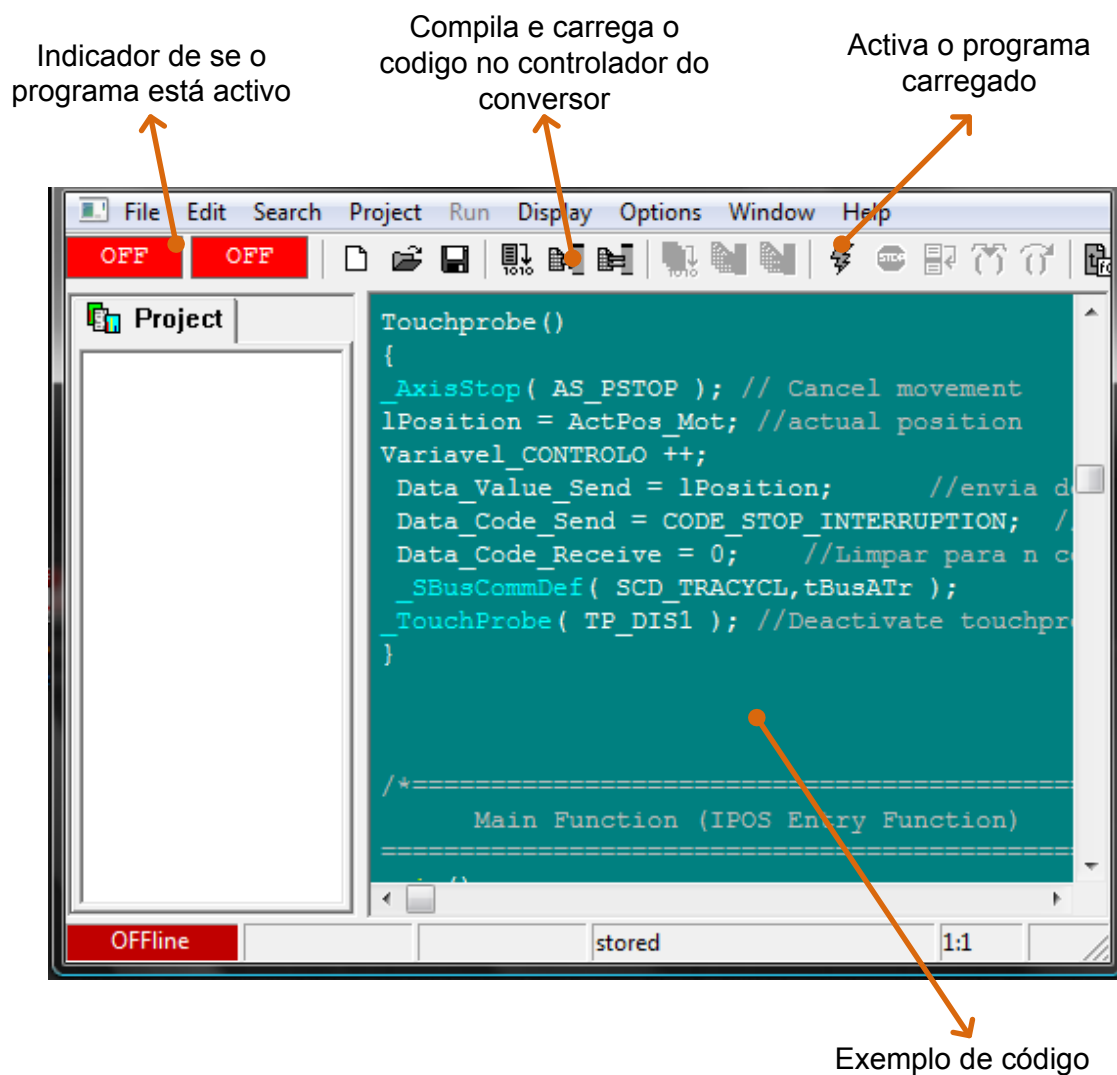



Figura 3.6: Compilador do MOVITOOLS V4.40 – Exemplo de como carregar um programa no controlador do conversor de frequência.

Depois de activo o programa, e se ainda estiver ligado ao PC, é possível verificar o seu comportamento, e controlar valores de variáveis. Mas para tal é necessário recorrer à janela da *Shell*, [Figura 3.7], e para termos acesso a ela, mais uma vez se pode consultar a Figura 2.7.

Antes de compilarmos e carregarmos programas no controlador, é necessário configurar o controlador para os valores próprios referente ao motor que controla, e isso faz-se precisamente na janela *Shell*. Para facilitar o trabalho é fazer as duas coisas de uma vez só, e para tal, criou-se 3 ficheiros com a extensão *mdx* que contem tanto o código assim como a configuração necessária para funcionar com os motores que se está a usar. Estes ficheiros são: ***ConfigMT_X.mdx***, ***ConfigMT_Y.mdx*** e ***ConfigMT_Z.mdx***; que correspondem ao motor referente ao eixo do X, do Y e Z respectivamente. Na Figura 3.7 apresenta-se uma ilustração visual de como aceder à janela onde se pode carregar as configurações para cada conversor. Depois de acedida, é só escolher a opção “copiar para o conversor o ficheiro de configuração” e confirmar.

copiar para o conversor o ficheiro de configuração

SHELL MOVITOOLS®

File Startup Parameters Display Extras Window Help

Main Menu

Parameter menus
0.. Display values
1.. Setpoints/ramp generators
2.. Controller parameters
3.. Motor parameters
4.. Reference signals
5.. Monitoring functions
6.. Terminal assignment
7.. Control functions
8.. Unit functions
9.. IPOS parameters
Applications
ApplicationBuilder - Applications

ONline | ADDR: 0 | Parameter saved

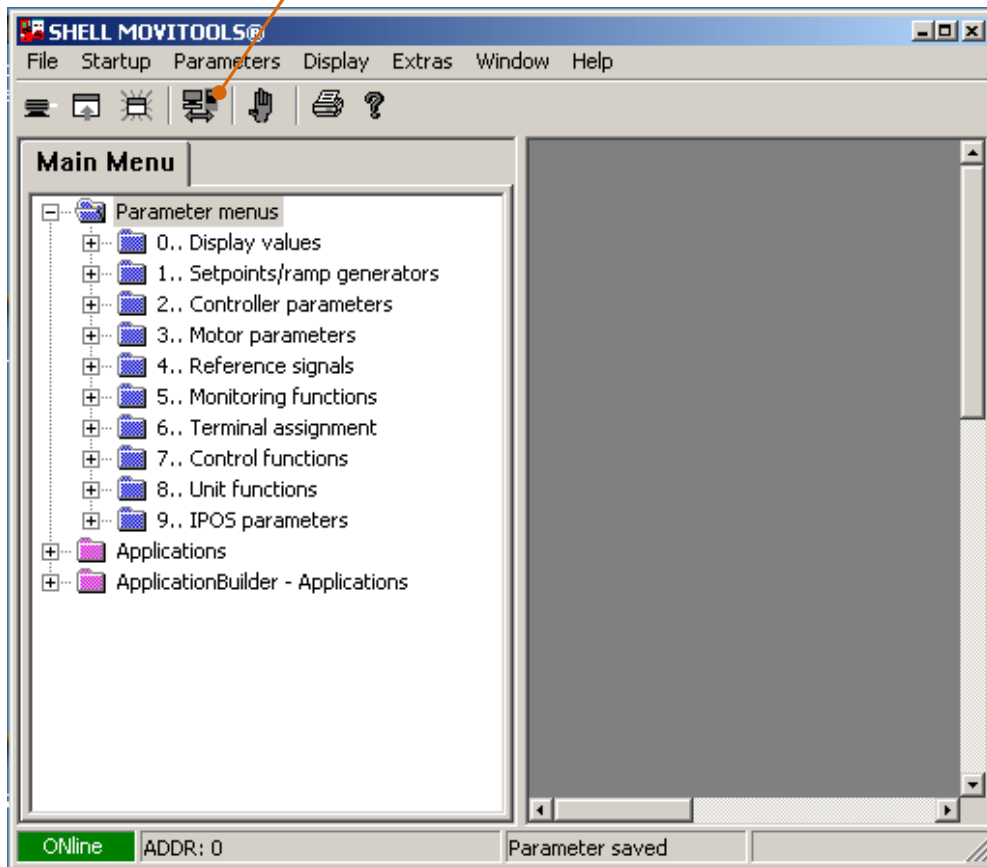



Figura 3.7: "Dentro da janela da Shell" - Indicação de como se pode aceder à janela onde se carrega as configurações criadas para cada conversor.

3.6 Montagem da bancada de ensaios

Nesta secção apresentam-se as imagens referentes à montagem efectuada em laboratório.

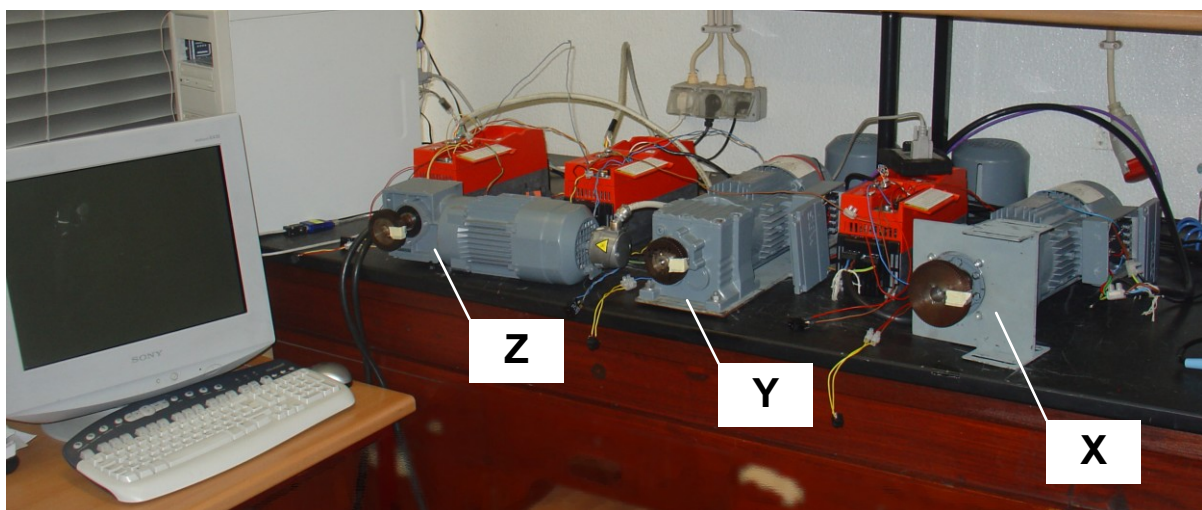



Figura 3.8: Perspectiva geral da montagem efectuada em laboratório

Na Figura 3.8, está representada a perspectiva geral da montagem da bancada de ensaios assim como quais os moto-redutores que se escolheram para cada um dos eixos.

Na Figura 3.9 pode-se identificar os dispositivos que se tem vindo a descrever, assim como a sua localização na montagem da bancada de ensaios.

Figura 3.9: Identificação dos principais componentes na montagem da bancada de ensaios

Foi com base nesta montagem que se desenvolveu o software-protótipo DACS-OPMG descrito no capítulo 4, assim como a resposta dos códigos carregados nos controladores dos conversores.

Numa primeira fase implementou-se o CAN-bus e testou-se a validade da comunicação entre nós, assim como a velocidade de sinalização. Isto é, testou-se a possibilidade de cada nó receber as suas mensagens e de enviar de volta, segundo uma velocidade de sinalização definida.

Depois testou-se a comunicação entre os conversores de frequência e os motores, assim como a possibilidade de receber valores pelo CAN-bus e de converter esses valores em posições ou velocidades.

Depois de interpretado e convertido correctamente os valores lidos no CAN-bus, passou-se para os testes com a variável de posicionamento para velocidades constantes de rotação, verificando-se assim as respostas dos encoders relativamente ás posições que se enviava. Foi nesta fase que se testou qual o numero de incrementos que cada encoder contou para que os redutores à saída completassem uma volta, verificando assim as relações apresentadas pelo fabricante na Tabela

2.1 e depois usadas na Equação 2.4 (ou seja $U_R = 28839$ e $U_{WAF} = 79475$) para calcular a interpolação linear.

Terminados os testes em relação ao posicionamento adicionou-se mais uma variável, a velocidade. Nesta fase, verificou-se novamente os valores fornecidos pelo fabricante [Tabela 2.1], e depois seguiu-se os testes em relação à interpolação linear, verificando assim que para distâncias diferentes percorridas, o tempo teria que ser o mesmo, ou seja os motores teriam de parar todos ao mesmo tempo variando assim a velocidade de modo a atingir essa concordância temporal entre eles.

Numa fase seguinte introduziu-se a leitura de ficheiros e a aplicação de todos os princípios testados até então, dando especial atenção à interpolação linear.

Para finalizar, testou-se ainda as interruções do IPOS®, usadas quer na calibração automática quer nas paragens de emergência.

4 Desenvolvimento do Software-Protótipo DACS-OPMG V1.0 e os códigos para os Conversores [1]

Cabe a este capítulo descrever então todos os códigos desenvolvidos para o controlo dinâmico dos moto-redutores da fresadora-protótipo descrita no capítulo 1 [PT103652, 2007][DOM756, 2007]. Como já foi referido, para se interagir com os conversores através do PC desenvolveu-se o software-protótipo DACS-OPMG V1.0, enquanto que do lado dos conversores, e para estes possam interpretar correctamente as ordens enviadas pelo DACS-OPMG, desenvolveu-se um código para cada um dos conversores. O DACS-OPMG foi programado em C++.NET, e para tal usou-se o Visual C++ 2008 Express Edition [Microsoft]. Os códigos para os conversores, por sua vez, foram escritos em C e compilados e carregados nos conversores com o Movitools v4.40 [SEW], como já tinha sido referido no capítulo 2.

O DACS-OPMG, tem este nome derivado do acrónimo para *Dynamic Axis Control System - Orthogonal Projection Motion Guide*. É um *Dynamic Axis Control System,* porque com este software é possível controlar 3 moto-redutores em tempo real (cada um associado a um eixo especifico no espaço 3D-XYZ). E o *Orthogonal Projection Motion Guide,* é uma representação gráfica da projecção ortogonal da trajectória descrita pelo movimento do ponto (onde estará a fresa) nos planos XY, XZ, e YZ, que tem como coordenadas as posições (já convertidas para unidades de comprimento) dos encoders.

Assim, começa-se por descrever o GUI do DACS-OPMG, e o modo de funcionamento deste do ponto de vista do utilizador, seguindo-se de uma descrição do código e dos objectos (mais concretamente os eventos associados a esses objectos) criados em C++.Net. Finalizando com uma descrição dos códigos carregados nos controladores.

4.1 GUI do Software.Protótipo DACS-OPMG V1.0

Organizou-se o GUI em grupos para facilitar a compreensão do mesmo [Figura 4.1]. O grupo "*Manual Control*" como o nome indica é onde o utilizador pode controlar manualmente e individualmente cada eixo e onde também recebe informação sobre cada eixo, tal como posição actual do encoder e se este se encontra em movimento ou não. O grupo "*External Input*" é onde se pode carregar ficheiros CAM desenvolvidos para este protótipo, e simulá-los ou enviá-los

---

[1] Para a leitura deste capítulo será necessário consultar Anexo A3, para se poder identificar a localização dos objectos no GUI de que se vai falar e o Anexo A4 para as variáveis Globais envolvidas.

directamente para a fresadora. Depois temos o grupo “*Global*” em que se pode enviar uma ordem de movimento para todos os eixos ao mesmo tempo, com base nos valores que estão definidos no grupo “*Manual Control*”, ou então pedir para actualizar as posições no programa com a posição real dos encoders. Neste grupo ainda podemos dar ordem a todos os motores para se movimentarem para uma posição inicial “*Home*”, efectuar uma nova calibração e limpar as projecções no OPMG. Para finalizar os grupos, temos o “Global IF ” responsável pelas inicializações e finalizações de processos e definição de propriedades de comunicação.

Todos estes grupos compõem o DACS, o OPMG por sua vez encontra-se representado à parte deste grupos e é, como já foi dito, onde se vai representar as projecções ortogonais [Figura 4.1].

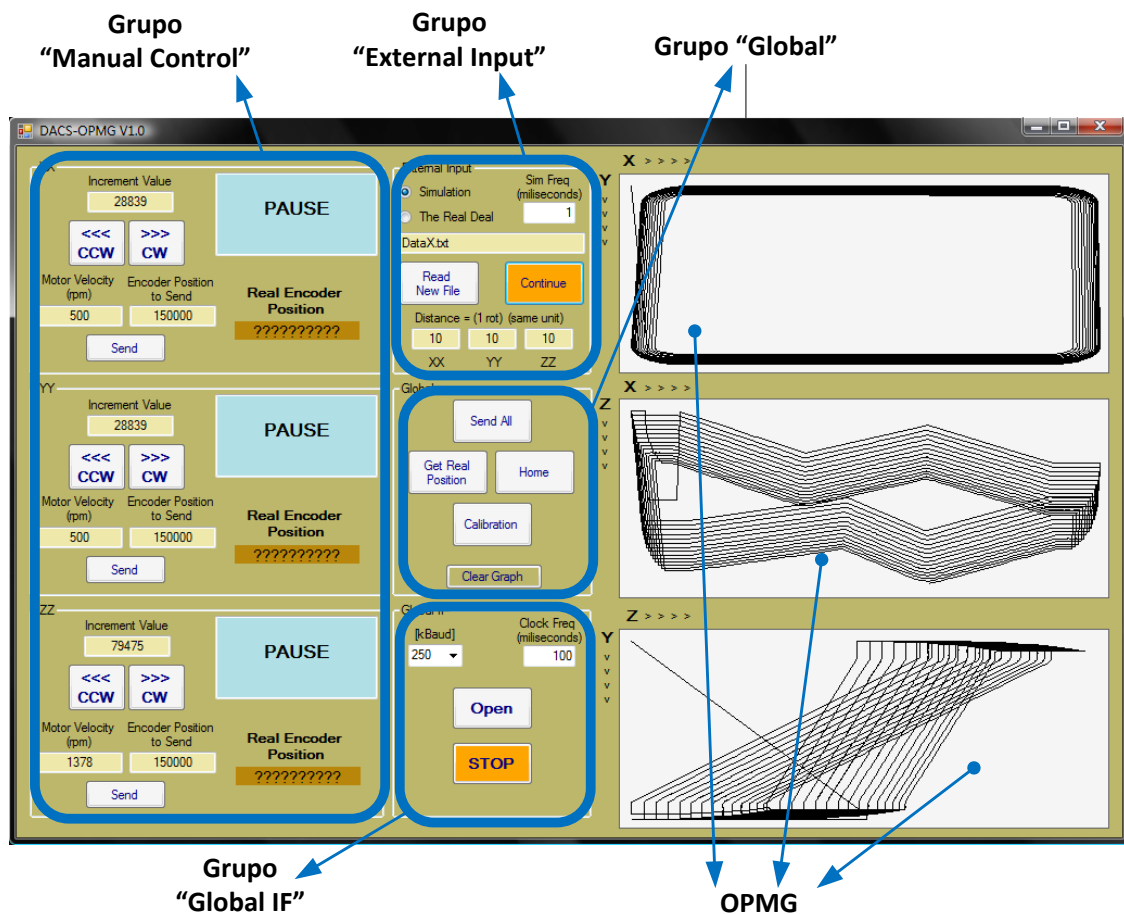



Figura 4.1: GUI do Software-Protótipo dividido por grupos.

4.1.1 GUI: Grupo “*Manual Control*”

Este grupo está dividido em 3 subgrupos “XX”, “YY” e “ZZ”, em que o subgrupo “XX” controla o moto-redutor convencionado para o eixo dos X, o “YY” para o eixo dos Y e claro o “ZZ” para o eixo dos Z.

O Subgrupo “XX”, apresenta três botões sendo eles representados no código pelos objectos *buttonGoBackX, buttonGoFrontX e buttonSendPositionX* [Anexo A3]. E para informação e input de

dados no programa, existem cinco caixas de texto representadas no código pelos objectos t*extBoxINincrementX*, *textBoxINspeedX*, *textBoxINPositionX*, *textBoxOUTlogX* e *textBoxOUTrealPositionX* [Anexo A3].

Assim, para fazer o motor rodar no sentido CW ou CCW, incrementando num sentido ou noutro as posições actuais do encoder, tem de se preencher o valor pretendido para o incremento em t*extBoxINincrementX*, e depois carregar no botão *buttonGoFrontX* para ele rodar no sentido CW ou então carregar em *buttonGoBackX* para rodar no sentido CCW. O valor preenchido no incremento é o valor real do encoder, isto é, é composto por valores inteiros positivos e cada valor corresponde $1/_{4096}$ de uma volta completa no eixo de rotação de saída do motor. A título de exemplo, se se preencher com valor 4096 o motor vai dar uma volta completa, mas para o utilizador observar à saída do redutor uma volta completa é preciso ter em conta as relações apresentadas na Tabela 2.1, o que numericamente significa que se tem de preencher com o valor28839.

Pode-se também enviar o motor para uma posição específica do encoder, preenchendo assim o *textBoxINPositionX,* com o valor pretendido, e depois carregar em *buttonSendPositionX.*

No que diz respeito às velocidades de rotação, com que as operações acima descritas vão, também podem ser controladas com este subgrupo para o eixo dos X, para tal, é necessário preencher mais uma vez com valores inteiros positivos menores ou iguais a 1500, o *textBoxINspeedX.* Este valor é interpretado pelo motor como sendo em rpm (rotações por minuto), e de igual forma não é essa a velocidade que o utilizador vê à saída do redutor, pois tem de se aplicar primeiro as relações apresentadas na Tabela 2.1.

Ainda neste subgrupo, existe o *textBoxOUTlogX*, que nos informa se o motor está em movimento ou não, ou se o ficheiro CAM chegou ao fim. E no *textBoxOUTrealPositionX*, é onde se pode ver a posição actual do encoder.

No que diz respeito ao funcionamento dos outros dois subgrupos o modo de operação é igual, divergindo apenas no motor alvo, onde mais uma vez é bom lembrar que podem apresentar relações diferentes das mencionadas no subgrupo “XX” [Tabela 2.1].

4.1.2 GUI: Grupo “*External Input*”

Este grupo é composto por 2 botões representados no código por *buttonReadStart* e *buttonReadCtn*, cinco caixas de texto representadas no código por *textBoxSIMclock*, *textBoxInpuFileRd*, *textBoxDisRotXX*, *textBoxDisRotYY* e *textBoxDisRotZZ* e ainda um par de botões de selecção, representados por *radioButtonSimulation e radioButtonRealDeal* [Anexo A3].

Neste grupo é possível simular um ficheiro CAM ou transferir as coordenadas lidas nesse ficheiro para os motores usando uma interpolação linear para garantir continuidade entre pontos.

Tal decisão é feita usando os botões de decisão, sendo que se o *radioButtonSimulation* estiver seleccionado, ao carregar no *buttonReadStart* este vai simular o ficheiro e apresentar as trajectórias no OPMG. Se por outro lado o *radioButtonRealDeal* estiver seleccionado, ao carregar no *buttonReadStart* inicia-se o processo real de fresagem. Ambos os processos carregam e lêem o ficheiro que estiver escrito no *textBoxInpuFileRd,* este ficheiro tem de estar na mesma directoria do executável, e tem de ser escrito com extensão.

No processo real são tidos em conta os valores preenchidos nos *textBoxDisRotXX*, *textBoxDisRotYY* e *textBoxDisRotZZ,* em que correspondem aos eixos dos X, Y e Z respectivamente. Estes valores representam as distâncias percorridas quando se obtêm uma volta completa à saída dos respectivos redutores. Estas distâncias devem ser preenchidas para cada redutor na mesma unidade em que os ficheiros CAM apresentam.

O *buttonReadCtn* apresenta dois estados, se tiver o texto "Pause" escrito, é responsável por colocar em pausa todo o processo de leitura quer no modo simulação, quer no modo real, mudando também o texto do botão para "continue". Se por outro lado, tiver o texto "continue" permite sair do estado de pausa e retomar o processo anterior mudando o texto para "pause".

4.1.3 GUI: Grupo "*Global*"

Este grupo é apenas constituído por botões, sendo eles representados em código pelos objectos *buttonSendAll*, *buttonGetRealPosition*, *buttonHome*, *buttonCalibration* e *buttonClearGraph* [Anexo A3].

O *buttonSendAll,* permite enviar em simultâneo para os três eixos as posições e velocidades preenchidas no grupo "*Manual Control*", descritas no sub capitulo 4.1.1.

O *buttonGetRealPosition* actualiza as posições actuais do encoder para cada eixo no grupo "*Manual Control*". Antes de iniciar um processo manual de movimento, convêm carregar neste botão para se saber onde se encontra o encoder, pois este ultimo é incremental e não absoluto, isto é, pode perder a posição em caso de corte de energia.

O *buttonHome,* envia a ordem a todos os motores para se movimentarem para o valor de calibração mais baixo obtido "Home" de cada eixo.

O *buttonCalibration* serve para calibrar a fresadora, isto é, primeiro é dada uma ordem de rotação no sentido CW para todos os motores até estes encontrarem o fim dos respectivos eixos, registando essa posição como sendo os máximos de calibração. Depois o processo é repetido no sentido oposto (CCW) e são registados os valores como sendo os mínimos de calibração. Finalizado estes dois processos os valores são guardados no ficheiro *ConfigCalibration.txt*, que se encontra na mesma directoria que o executável.

Os motores param nas extremidades quando activam as interrupções detectadas pelo IPOS® representadas na montagem da Figura 3.4.

Por fim, o *buttonClearGraph* limpa os gráficos das projecções ortogonais, para se descobrir onde se encontra a fresa naquela momento.

4.1.4 GUI: Grupo "*Global IF*"

Este último grupo é composto por dois botões, sendo eles representados em código pelos objectos *buttonOpenClose* e *buttonSTOP*, uma caixa de texto representada pelo *textBoxClockFreq* e uma "Combo Box" representada pelo *comboBoxBaud* [Anexo A3].

O *buttonOpenClose* é o responsável por abrir uma comunicação com o CAN-bus através do CANUSB e fechar [CANUSB]. Este botão apresenta assim dois estados, "Open" ou "CLose". Se tiver com o texto "Open" então abre um protocolo de comunicação com o CAN-bus, e activa todos os botões para que se possa iniciar os trabalhos. Se por outro lado o texto estiver com "Close", fecha essa porta de comunicação e desativa os botões. Assim sendo este botão é aquele que inicia todo o processo pelo quais os outros enviam mensagens, logo convêm abrir um protocolo de comunicação antes de iniciar qualquer tipo de comunicação com os conversores. Pode-se contudo fazer simulações de ficheiros sem abrir um canal de comunicação através deste processo.

O *buttonSTOP,* é aquele que pára todos os motores em caso de necessidade, mesmo que estes se encontrem a meio de um processo de leitura de ficheiro. Assim sendo este botão uma vez aberta uma comunicação fica sempre activo para emergências e afins.

No *textBoxClockFreq* preenche-se o valor em milissegundos que vai determinar com que rapidez corre a máquina de estados associada ao evento do timerIO, descrito mais à frente. Em caso de erros de comunicação este valor pode ser aumentado para procurar erros.

Por fim no *comboBoxBaud* é dada a possibilidade de escolher valores predefinidos para o "baud rate", em que o valor $250\ \mathrm{kBaud}$ é o que se encontra escolhido por defeito em ambos os extremos do CAN-bus. Uma alteração deste valor implica alterar também nos conversores.

4.1.5 GUI: OPMG

Como já foi referido o *Orthogonal Projection Motion Guide,* é uma representação gráfica da projecção ortogonal da trajectória descrita pelo movimento da fresa nos planos XY, XZ, e YZ. Assim o que se visualiza nestes gráficos é o movimento da fresa ao longo do processo de fresagem e não a projecção da figura geométrica que se irá obter no final da fresagem. Assim quando a fresa se mexe por consequência do movimento dos motores em cada eixo essa trajectória é desenhada no OPMG

como uma interpolação linear do movimento de um ponto inicial para um ponto final. A origem desses gráficos encontra-se no canto superior esquerdo.

4.1.6 GUI: Extras

De sempre que se usa o programa e este dá ordem de movimento ou se simula um ficheiro, é criado na directoria do executável um registo de eventos, em que o nome desse ficheiro é diferente para cada hora de utilização. O ficheiro tem a extensão txt, e inicia-se sempre com a palavra LOG seguindo-se da data e hora, por exemplo, para o dia 14-09-2009 ás 17 horas o ficheiro criado tem o nome de *LOG-2009-09-14-17.txt*. Lá dentro pode-se encontrar pontos de controlo por onde o programa passou enquanto corria, assim como erros que possam ter acontecido. Útil para verificar se algum processo funcionou como devido por parte do programa.

4.2 Eventos associados aos objectos criados no DACS-OPMG

Para se perceber, como cada um dos objectos referidos anteriormente e representados no [Anexo A3], funciona em termos de código desenvolvido, criou-se um conjunto de fluxogramas para ajudar no processo de compreensão.

Assim o primeiro fluxograma [Figura 4.2], representa o evento associado ao carregar no botão referente ao objecto *buttonOpenClose.* Neste fluxograma pode-se observar como o botão muda entre os dois estados associados ao botão (“Open” e “CLose”), que erros pode produzir e quais as variáveis globais que altera ou usa. As variáveis globais encontram-se no ficheiro Global.h do projecto, e sempre que exista alguma alteração nelas ou se aceda a elas, é possível ver no fluxograma, pois está escrito no canto superior direito das caixas referentes à manipulação de dados (paralelogramos azuis) [Anexo A.4].

O fluxograma que se segue [Figura 4.3], representa os eventos associados ao carregar dos botões, *buttonGoBackX*, *buttonGoFrontX*, *buttonGoBackX*, *buttonGoFrontX* e *buttonGoBackX*, *buttonGoFrontX.* Decidiu-se representar todos no mesmo fluxograma, porque formalmente são iguais mudando apenas o conteúdo. Assim escolheu-se representar como modelo o evento para o objecto *buttonGoBackX,* e para se obter o fluxograma dos outros objectos, basta alterar no fluxograma os caracteres a vermelho pelos caracteres representados no topo (dentro dos comentários com o nome desse objecto).

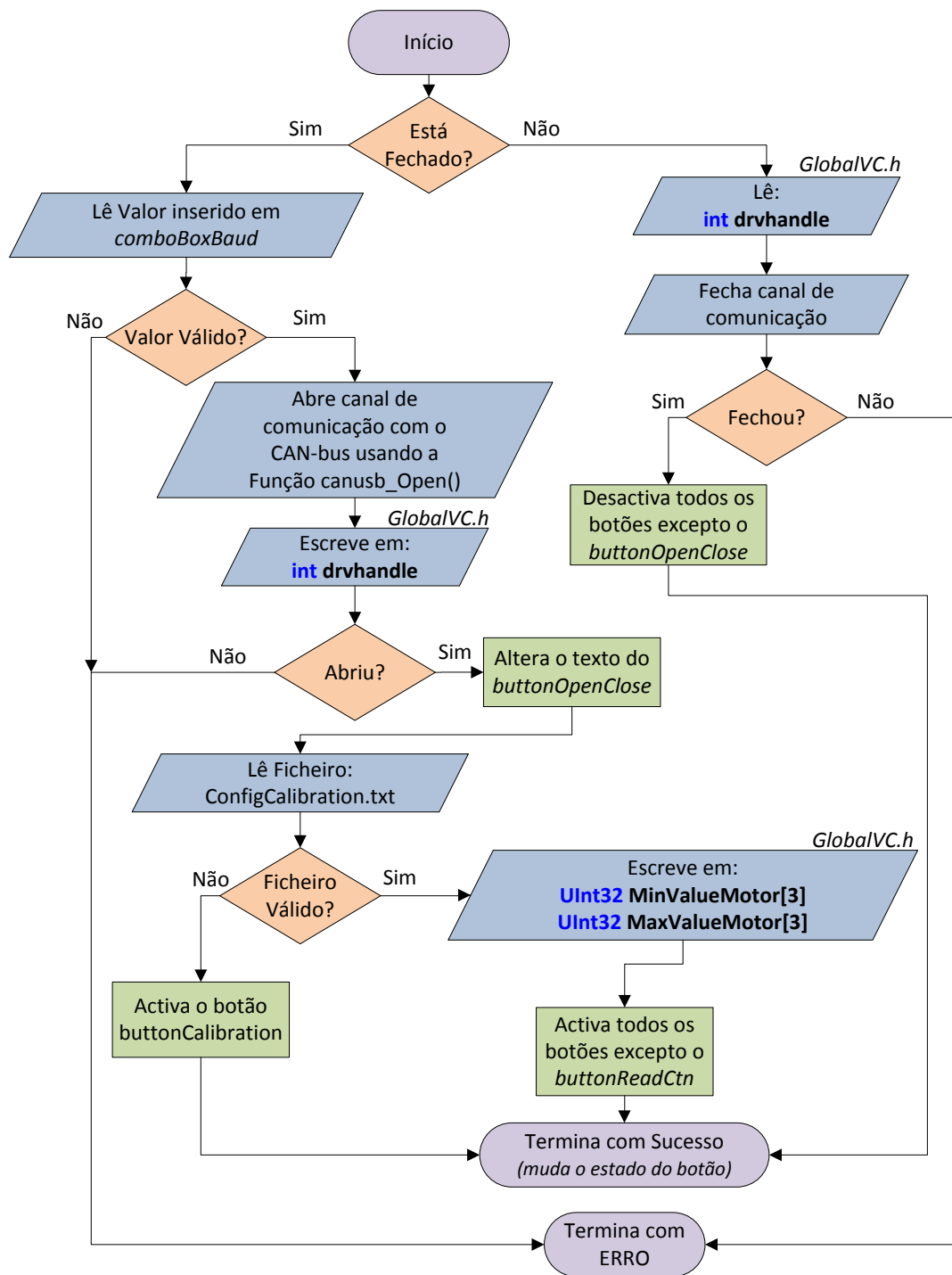



Figura 4.2: Fluxograma para o evento associado ao "click" do objecto buttonOpenClose

Olhando ainda para a Figura 4.3, pode-se verificar que se o evento for concluído com sucesso, isto é se carregar os valores referentes à velocidade e posição nas variáveis globais correspondentes [Anexo A.4], é activado a máquina de estados descrita no subcapítulo 4.3 passando imediatamente para o estado com o nome SEND_DATA_SPEED, também explicado mais à frente no subcapítulo 4.3, que vai dar inicio o processo para movimentar o motor alvo.

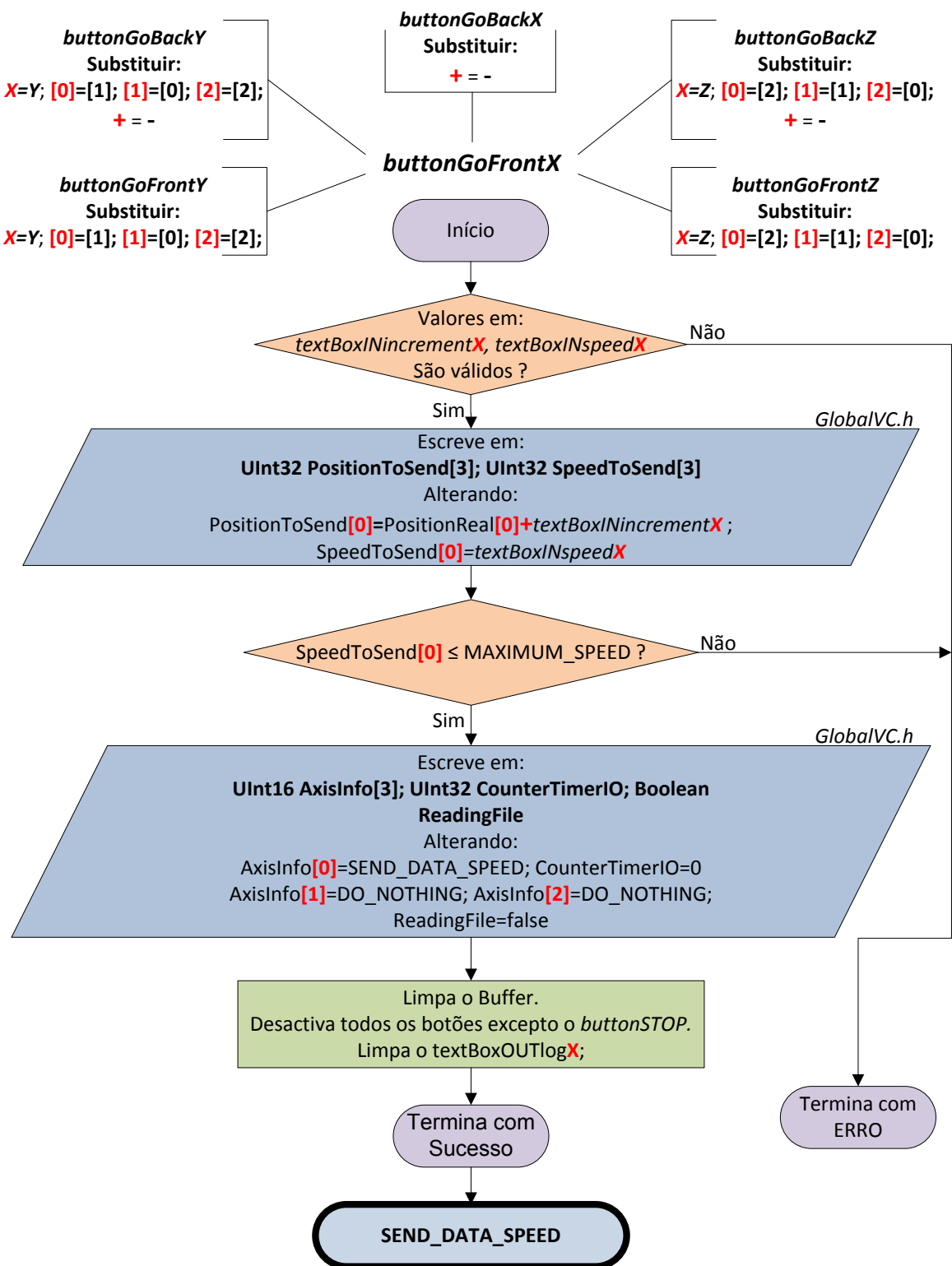



Figura 4.3: Fluxograma para os eventos associados ao "click" dos objectos buttonGoBackX, buttonGoFrontX, buttonGoBackX, buttonGoFrontX e buttonGoBackX, buttonGoFrontX.

Na Figura 4.4, o processo é idêntico ao da Figura 4.3 diferindo apenas no carregamento dos valores para a posição, enquanto que na Figura 4.4, é carregado o valor directamente da caixa de texto associada, na Figura 4.3 é feito um incremento à posição real e só depois carregado o valor

para a posição. De igual modo a figura Figura 4.4 apresenta os eventos associados aos *buttonSendPositionX*, *buttonSendPositionY* e *buttonSendPositionZ.*

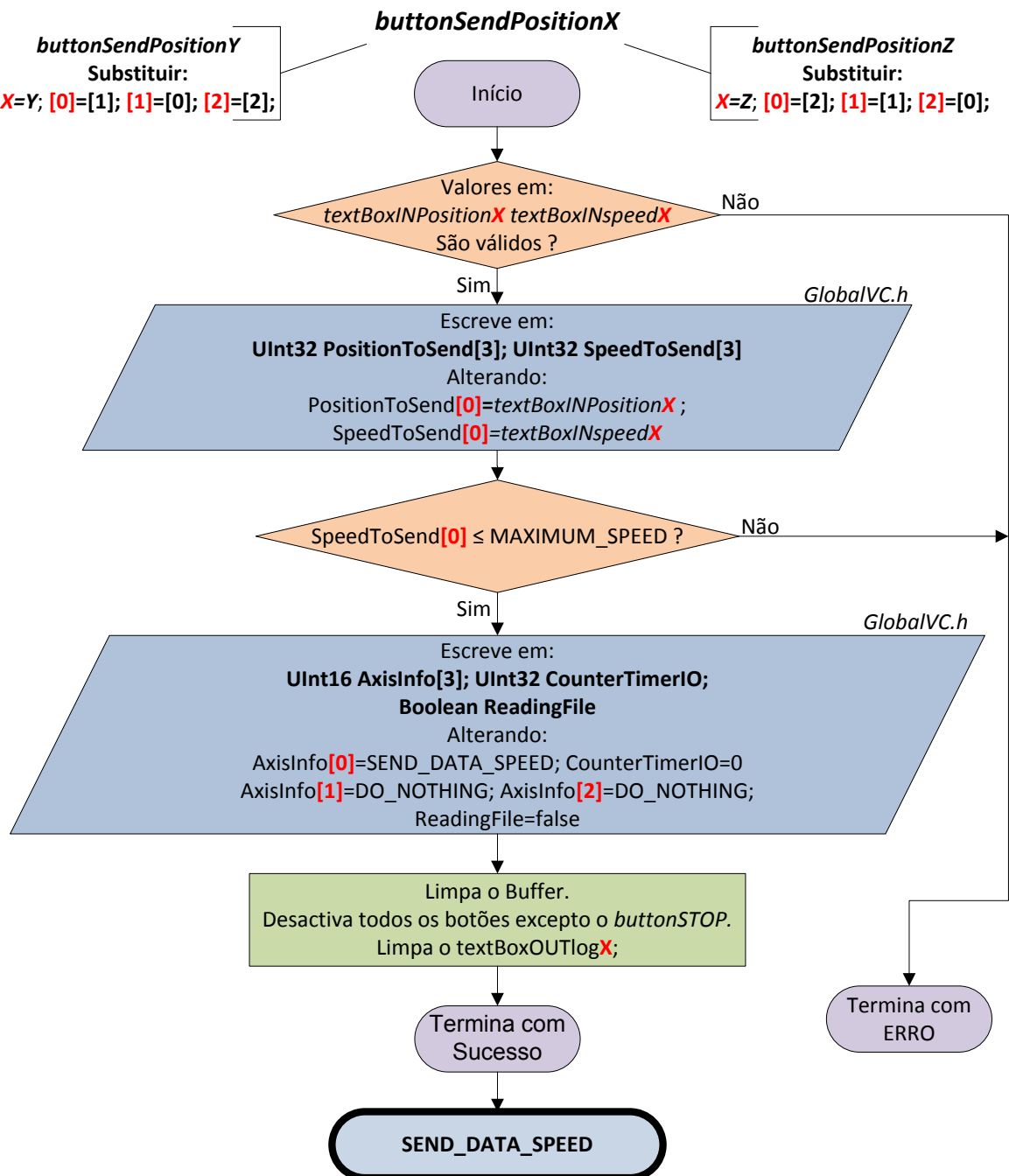



Figura 4.4: Fluxograma para os eventos associados ao "click" dos objectos buttonSendPositionX, buttonSendPositionY e buttonSendPositionZ.

O fluxograma representado na Figura 4.5, é na sua forma idêntico ao da Figura 4.4, com a diferença de que não carrega apenas os valores para um eixo mas sim para todos os eixos, e representa o evento associado ao buttonSendAll. E mais uma vez se terminado com sucesso vai activar a máquina de estados e iniciar o processo de movimento dos motores.

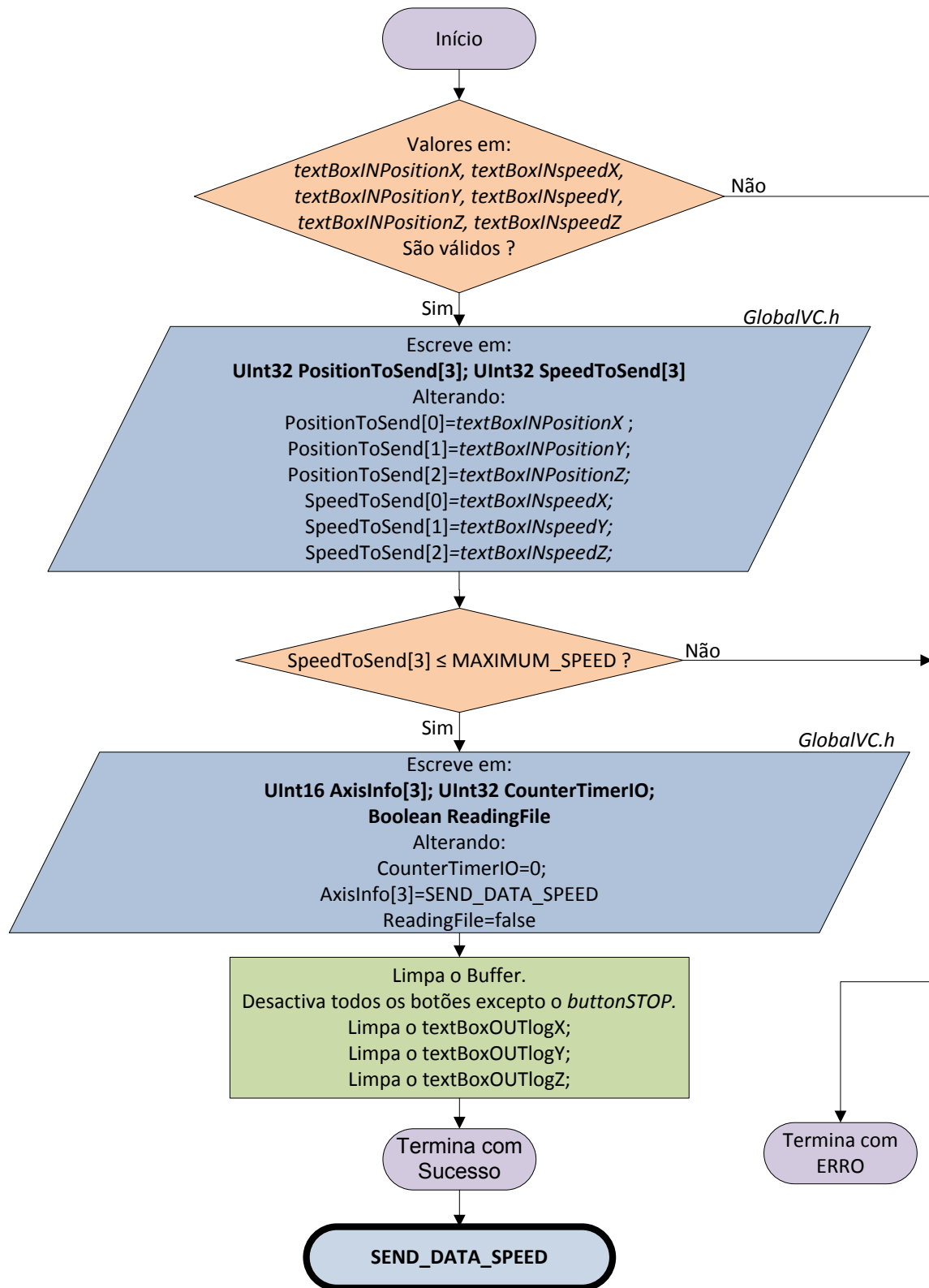



Figura 4.5: Fluxograma para o evento associado ao "click" do objecto buttonSendAll

Na Figura 4.6 temos o fluxograma para o evento associado ao objecto *buttonHome,* e neste caso não existe a possibilidade de terminar com erro por parte do utilizador, pois os valores são carregados internamente, passando logo para a máquina de estados.

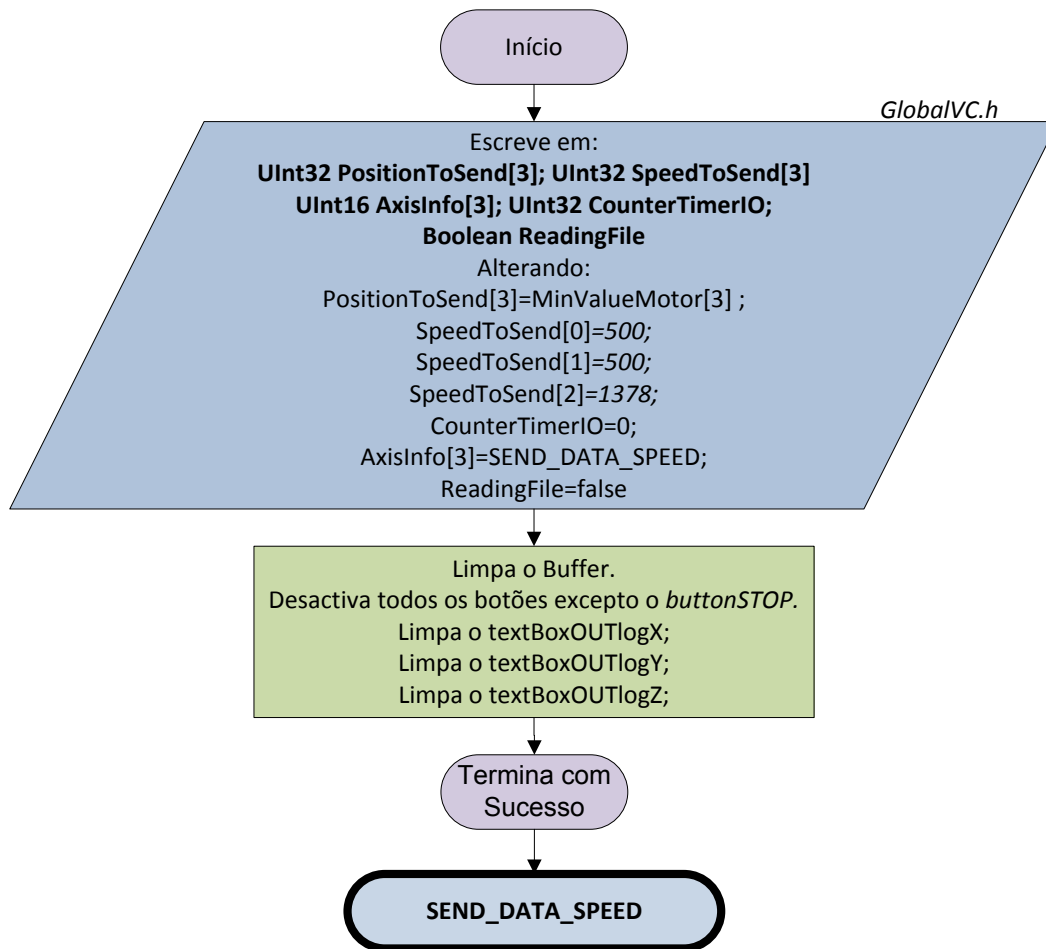



Figura 4.6: Fluxograma para o evento associado ao "click" do objecto buttonHome

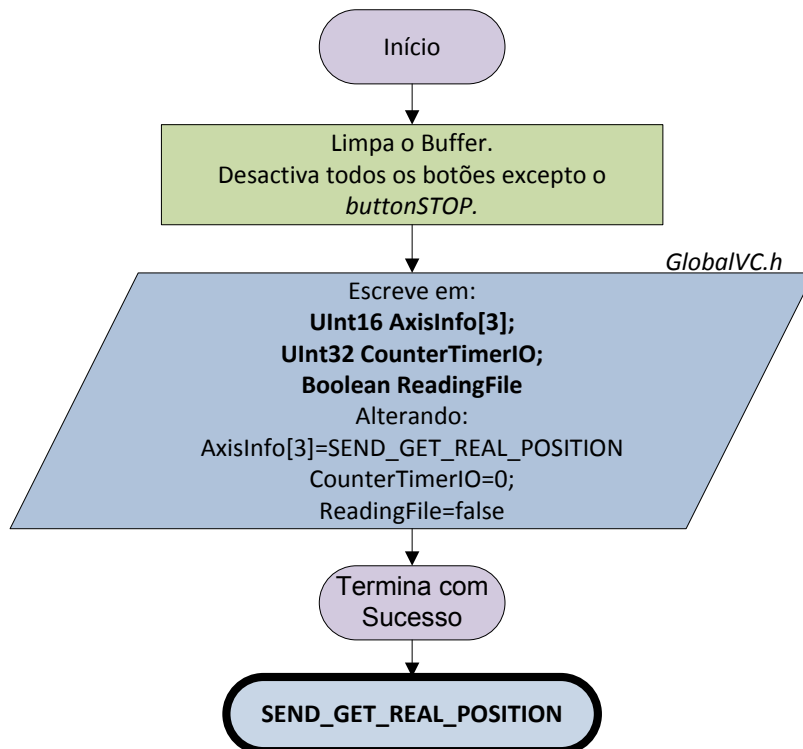



Figura 4.7: Fluxograma para o evento associado ao "click" do objecto buttonGetRealPosition

Na Figura 4.7, temos o fluxograma para o evento associado ao *buttonGetRealPosition*, neste evento, os valores são carregados internamente como o descrito na Figura 4.6, no entanto, neste caso é iniciado a máquina de estados num estado diferente dos anteriores o estado SEND_GET_REAL_POSITION, descrito mais à frente em detalhe no subcapítulo 4.3. Então este processo não vai activar o movimento dos motores mas sim pedir a posição real dos encoders.

O fluxograma representado pela Figura 4.8 é idêntico ao da Figura 4.6, carrega também os valores internamente antes de iniciar o processo de ordem de movimento dos motores através da máquina de estados, mas carrega "informadores" diferentes [Anexos A.4], para levar a cabo a calibração.

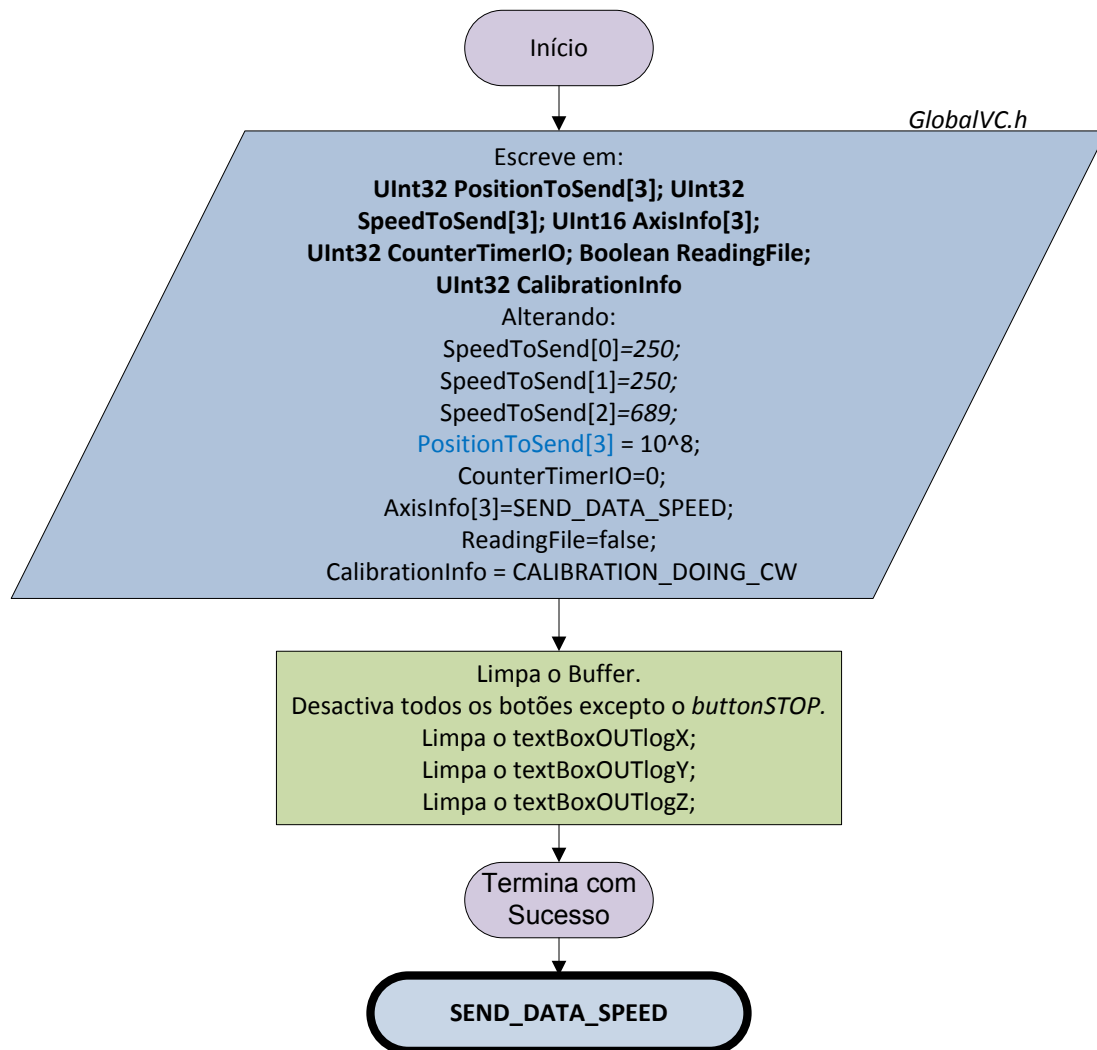



Figura 4.8: Fluxograma para o evento associado ao "click" do objecto buttonCalibration

Olhando agora para a Figura 4.9, podemos ver que temos um evento diferente dos anteriores. Este evento corresponde ao *buttonReadStart*, responsável por iniciar a leitura dos ficheiros CAM, como já tinha sido dito. O resultado depende neste caso do estado em que se encontrar o botão de selecção, representado por *radioButtonSimulation* [Anexo A3]. Assim sendo,

existe duas possibilidades, ou inicia a máquina de estados no estado INITIALIZATION_FILE (descrito mais à frente - subcapítulo 4.3), inicializando assim o ciclo de leitura e movimento de motores, ou então simula o ficheiro usando o *timerSIM* (tambem explicado no subcapítulo 4.4, mais em detalhe), que serve apenas para simular o ciclo de leitura à parte sem interferir com a máquina de estados.

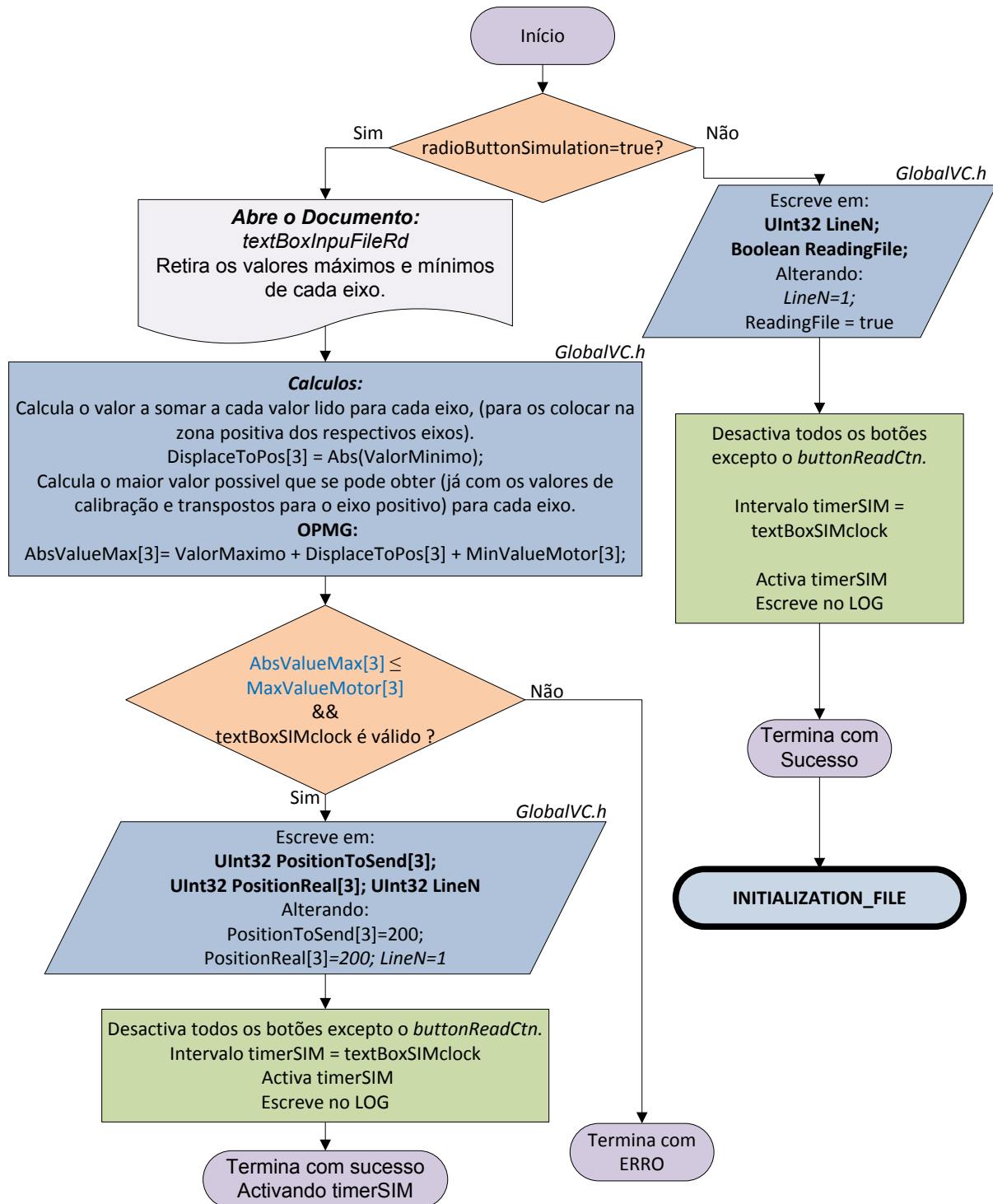



Figura 4.9: Fluxograma para o evento associado ao "click" do objecto buttonReadStart

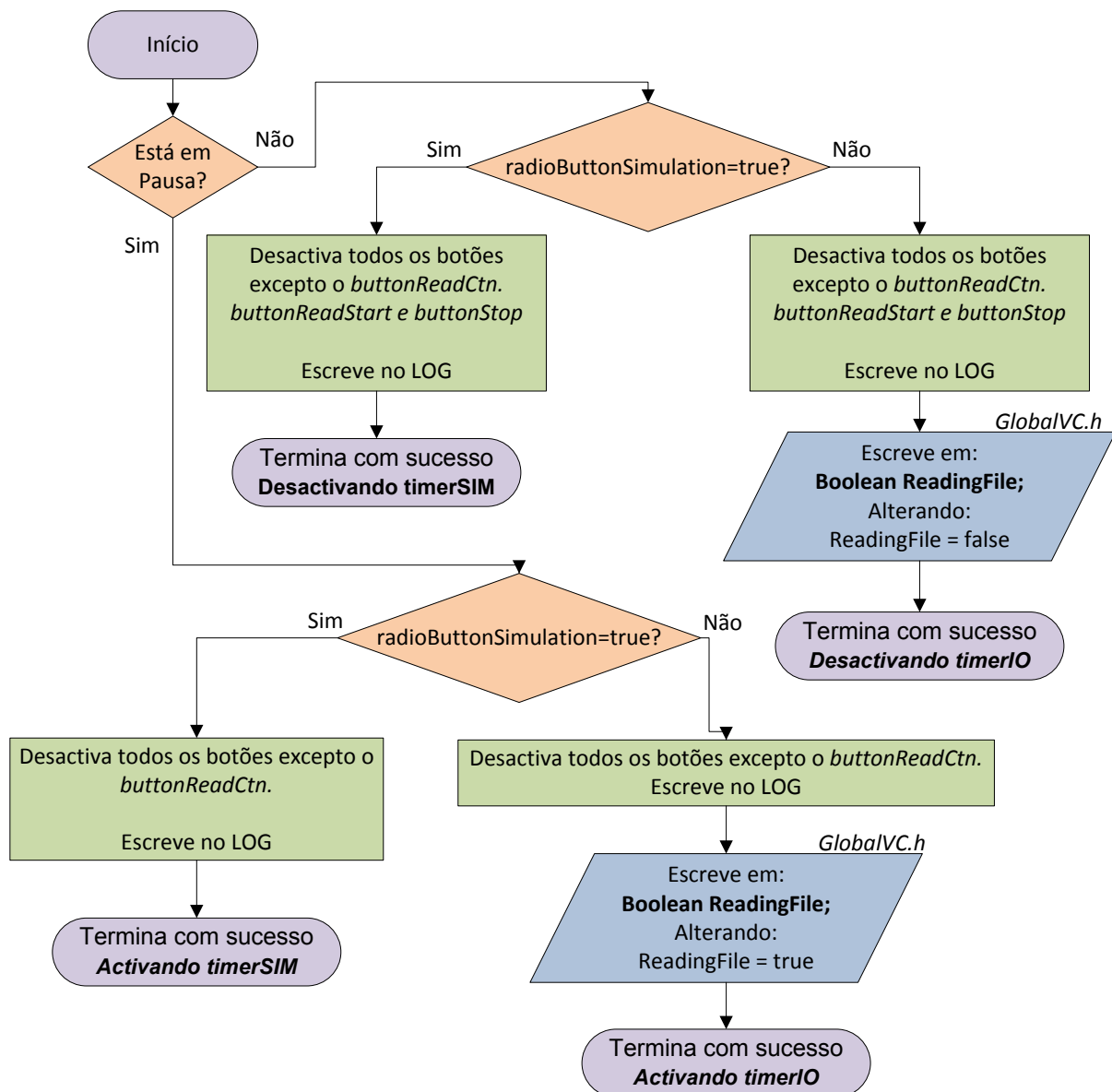



Figura 4.10: Fluxograma para o evento associado ao "click" do objecto buttonReadCtn

Na Figura 4.10, temos o fluxograma representativo do evento associado ao *buttonReadCtn*, este evento terá quatro finais possíveis, se o botão se encontrar no estado de "pausa", ele dependendo do *radioButtonSimulation* vai desactivar o *timerIO* ou o *timerSIM*, contudo, este desactivar não passa disso mesmo um desactivar, porque os estados onde se encontram não são alterados, e assim, no próximo "click" o botão vai estar no estado "continuar" e vai activar os timers nos mesmos estados em que estavam.

Finalmente na Figura 4.11, temos o evento associado ao carregar no botão *buttonSTOP*. Este evento também depende como os dois anteriores do *radioButtonSimulation,* e dependendo disso ou termina por completo o timerSIM ou muda o estado na máquina de estados no timerIO para o estado SEND-STOP, que depois irá parar os motores.

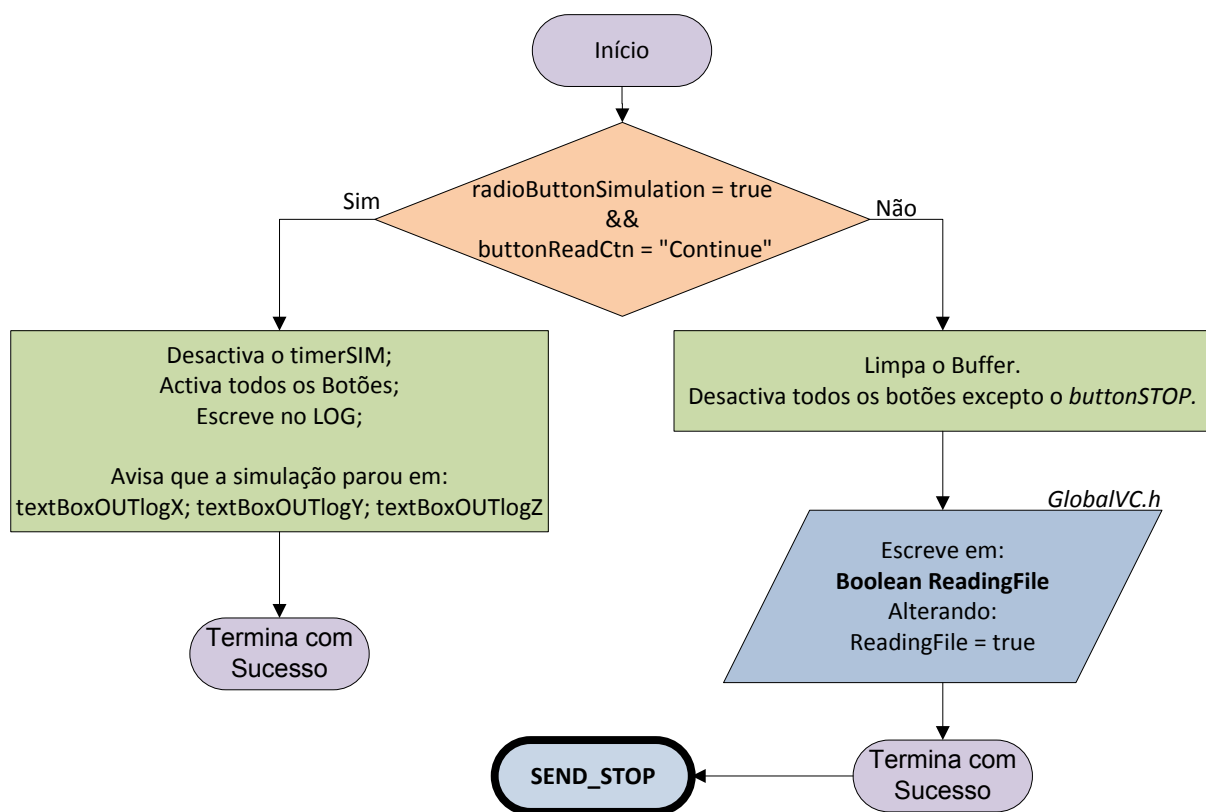



Figura 4.11: Fluxograma para o evento associado ao "click" do objecto buttonSTOP

4.3 Máquina de Estados associada ao Evento do *timerIO*

No subcapítulo anterior viu-se que os eventos terminavam muita das vezes com a inicialização da máquina de estados num determinado estado. Ora bem, essa máquina de estados corre dentro do evento *timerIO*, com uma frequência determinada pelo objecto *textBoxClockFreq*. Olhando para Figura 4.12, pode-se ver que o *timerIO* é inicializado pelo utilizador (através dos eventos explicados anteriormente) em estados específicos tais como o SEND_DATA_SPEED, SEND_GET_REAL_POSITION, INITIALIZATION_FILE e o SEND_STOP. Depois desta inicialização existem estados sequenciais associados aos estados referidos, que flúem apenas num sentido sequencial, ou seja, se tudo correr bem apenas têm o estado seguinte na cadeia como destino. Para um melhor entendimento do funcionamento de cada estadp, criou-se diagramas SDL para cada um deles. Mais uma vez se relembra que para uma melhor compreensão dos diagramas é necessário a consulta dos Anexo A.3 e Anexo A.4. Cada diagrama contém uma legenda, e nessa legenda existe um ponto verde que serve para indicar quando se escreve no LOG [subcapítulo 4.1.6].

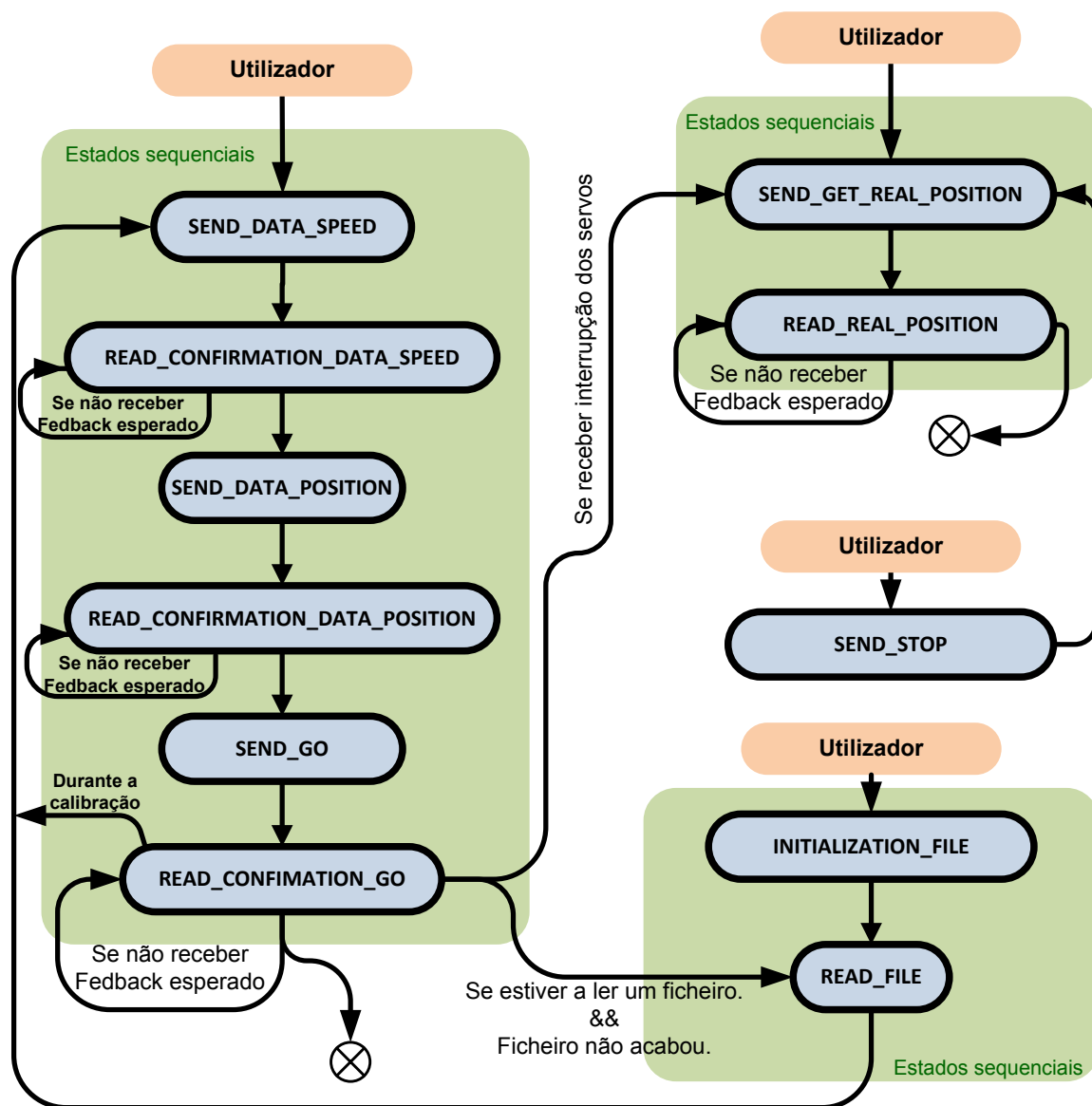



Figura 4.12: Máquina de estados associada ao evento timerIO e as relações possíveis entre estados

Na Figura 4.13, temos a descrição do evento associado ao estado SEND_DATA_SPEED, este evento decide com base no "informador AxisInfo" [Anexo A.4] para quais motor deve enviar as velocidades e passa para o estado READ_CONFIRMATION_DATA_SPEED [Figura 4.14]. Neste estado o CAN-bus é lido com uma frequência imposta pelo *timerIO*, até receber a confirmação de todos os conversores para os quais enviou a velocidade, assim recebidas todas as confirmações passa para o estado seguinte. Mas se por alguma razão não houver confirmação de alguns dos conversores que devia responder de volta, a máquina de estados permanecesse neste estado por um período determinado (neste caso 5 segundos) continuando a procurar por uma resposta, terminado esse período a máquina de estados termina com erro.

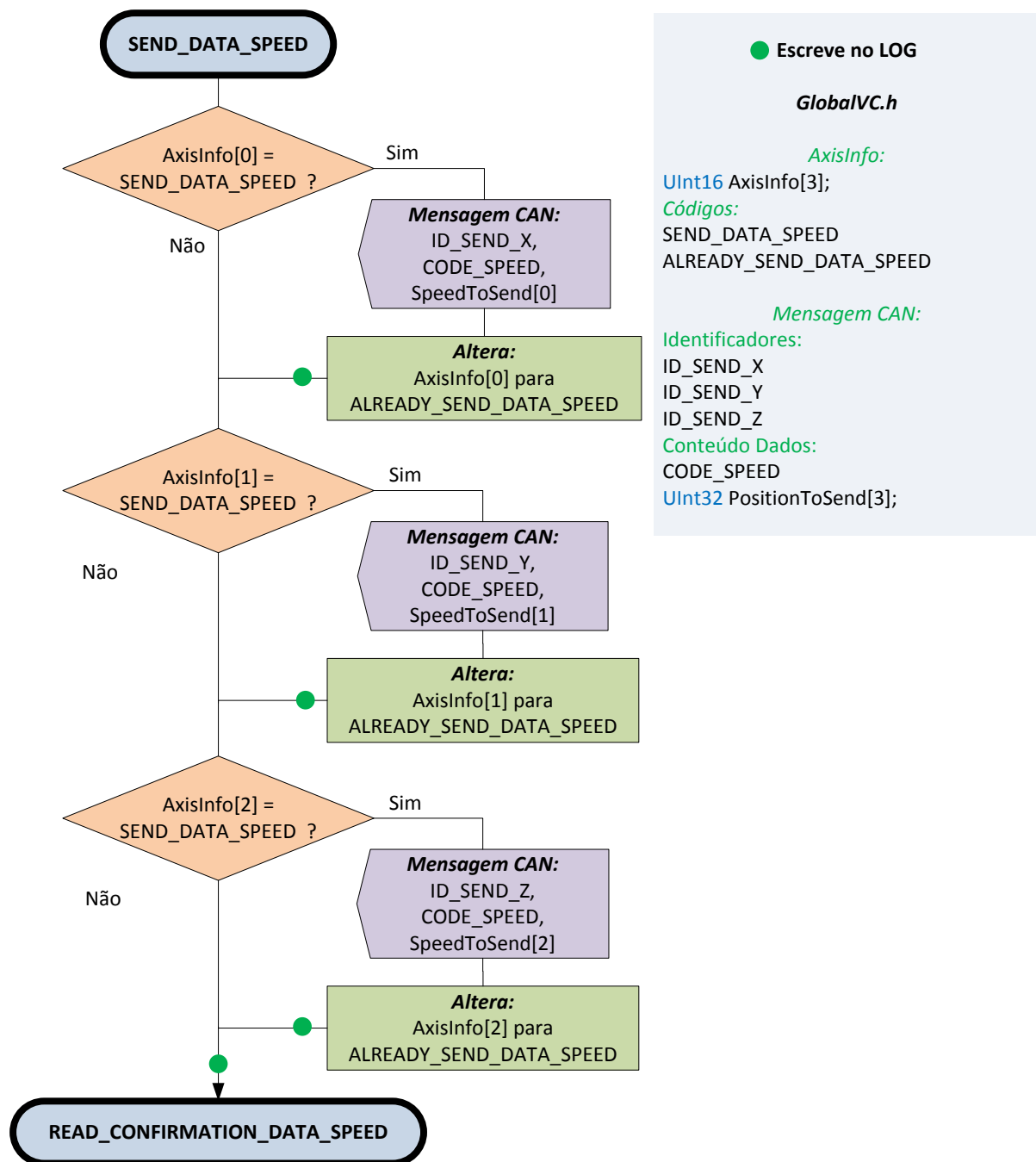



Figura 4.13: Diagrama SDL para o Estado SEND_DATA_SPEED.

O estado que se segue no caso de haver confirmação de todos os conversores é o SEND_DATA_POSITION [Figura 4.15]. Este estado é semelhante ao estado SEND_DATA_SPEED [Figura 4.13], excepto no conteúdo do “informador AxisInfo” e nos valores enviados, que em vez de ser velocidades são posições.

O estado que se segue logo é o READ_CONFIRMATION_DATA_POSITION [Figura 4.16] que também é bastante semelhante ao estado READ_CONFIRMATION_DATA_SPEED [Figura 4.14], mudando de igual modo apenas o conteúdo do “informador AxisInfo” que é próprio de cada estado.

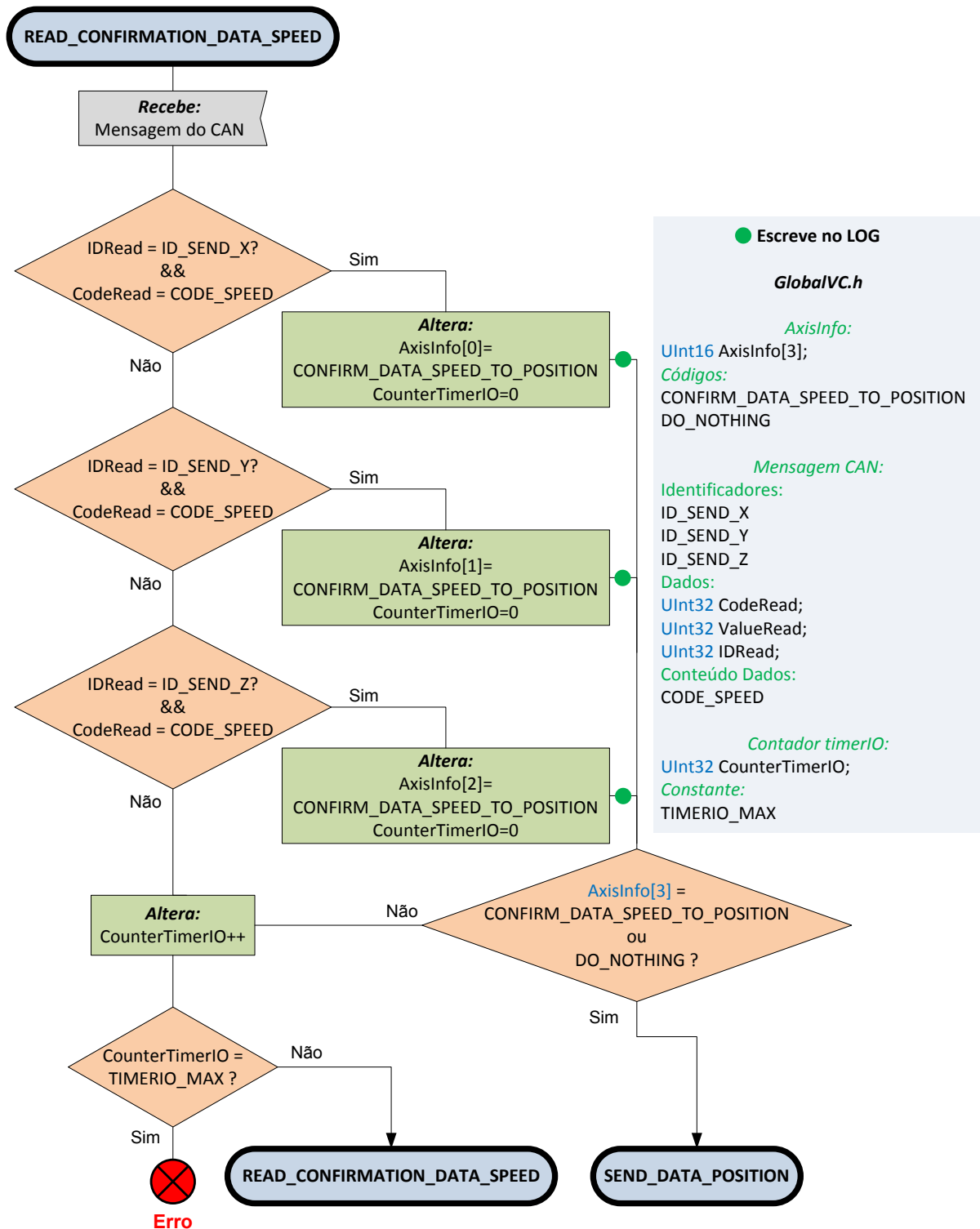



Figura 4.14: Diagrama SDL para o Estado READ_CONFIRMATION_DATA_SPEED.

Ao receber a confirmação de todos os conversores para os quais enviou valores, dá-se por concluído o processo de carregar os valores nos conversores, a partir deste ponto os conversores esperam pela confirmação para iniciarem o movimento nos motores com base nesses valores carregados (velocidade e posição).

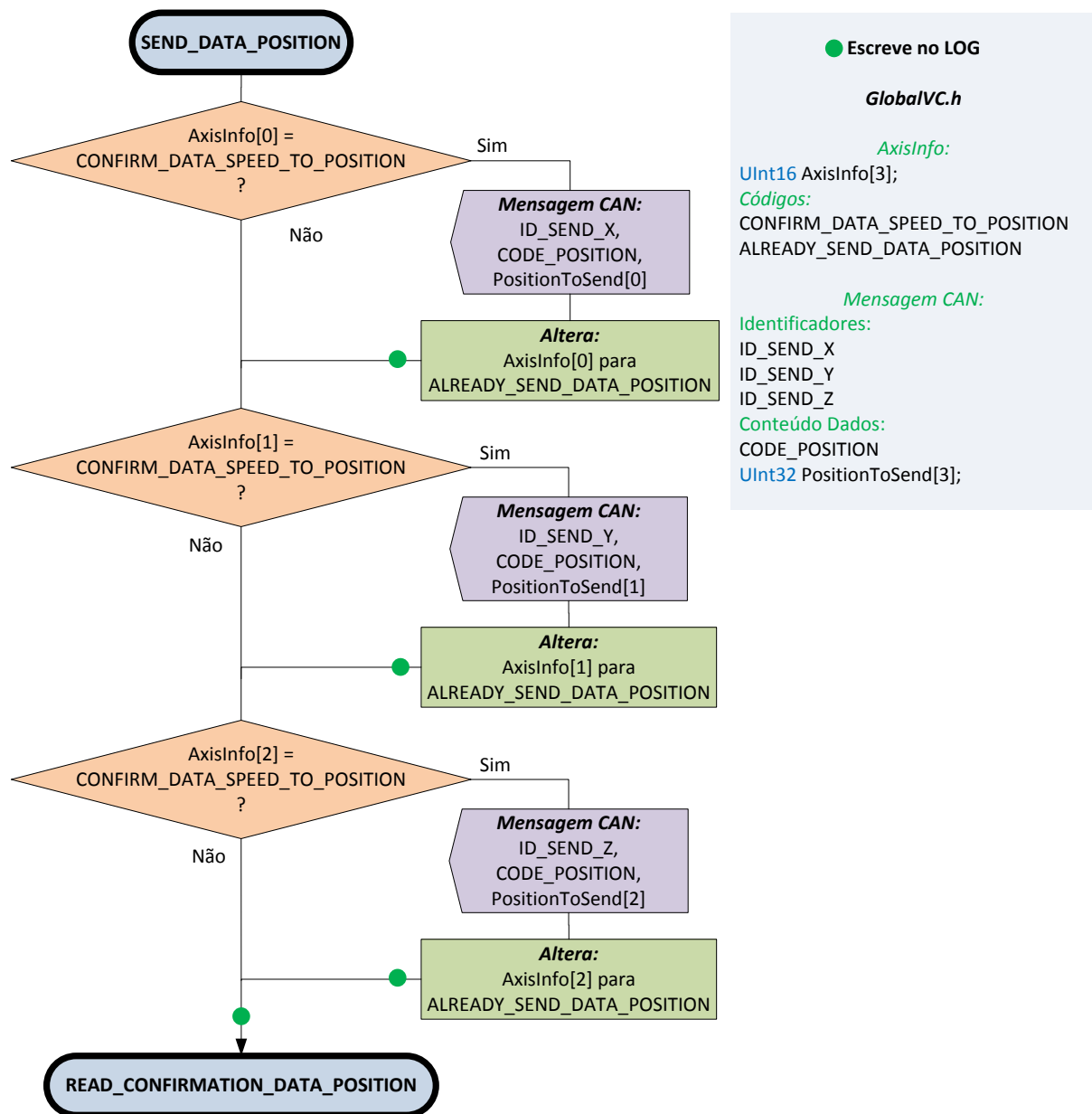



Figura 4.15: Diagrama SDL para o Estado SEND_DATA_POSITION.

Então o estado que se segue é o SEND_GO [Figura 4.17], que envia então para os conversores com os quais tem comunicado nos estados anteriores, a confirmação para iniciarem o movimento e passa logo para o estado READ_CONFIRMATION_GO [Figura 4.18][Figura 4.19].

É neste estado que se permanece aquando o movimento dos motores, assim que os motores estiverem em movimento enviam mensagens com essa indicação na rede CAN-bus que é intrepretada por este estado. Também é neste estado que se verifica se o processo está em modo de calibração, e se assim for este estado inicia todo o processo descrito até aqui novamente, enviando novos dados para se efectuar a segunda parte da calibração no sentido CCW. E depois de terminado o processo de calibração, é neste estado que se verifica isso e se reencaminha para o estado SEND_GET_REAL_POSITION [Figura 4.20].

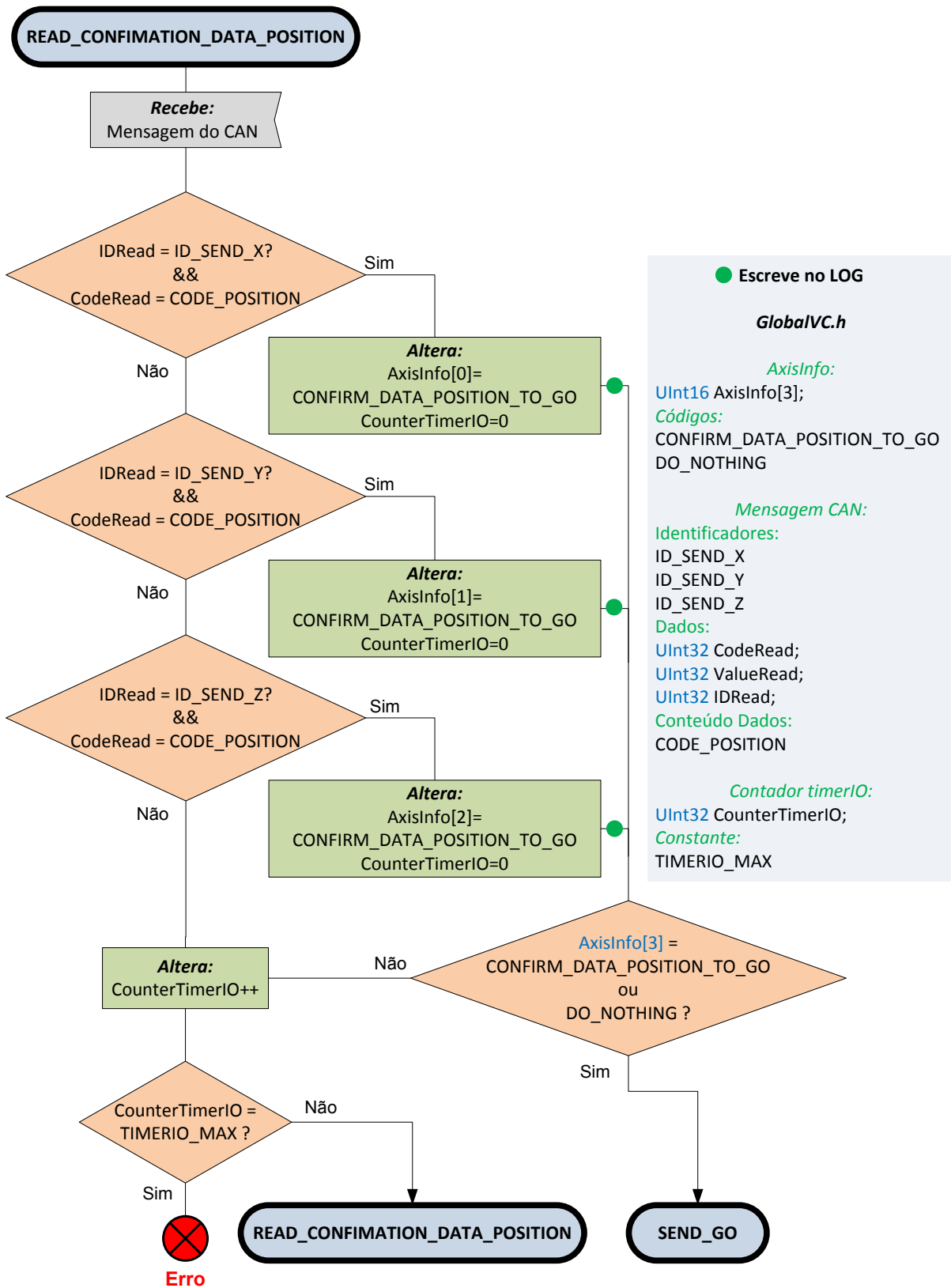



Figura 4.16: Diagrama SDL para o Estado READ_CONFIRMATION_DATA_POSITION

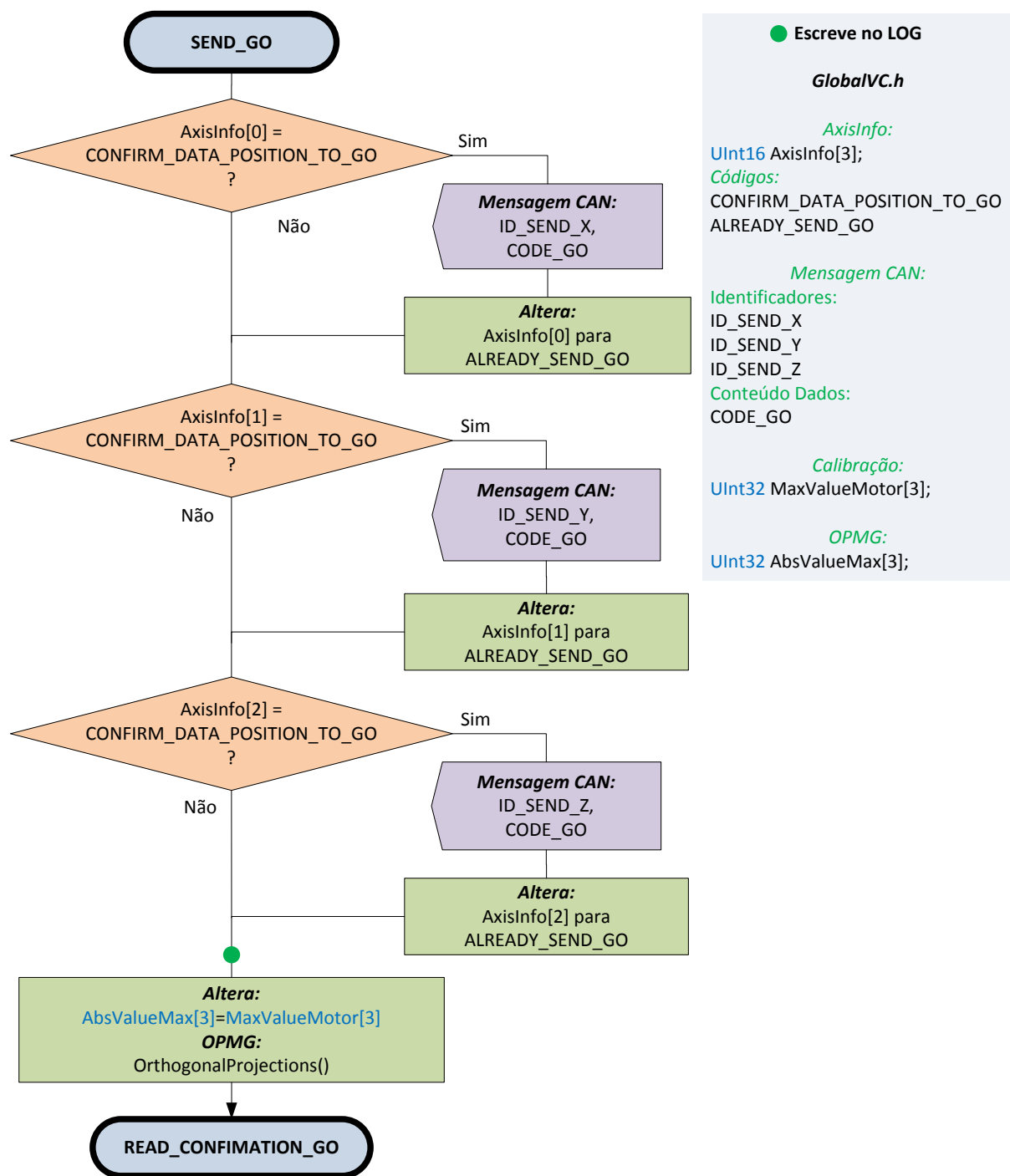



Figura 4.17: Diagrama SDL para o Estado SEND_GO.

Ainda no estado READ_CONFIRMATION_GO, é onde se verifica, depois de finalizado um processo de movimento nos motores se se está a ler um ficheiro CAM ou não, e em caso afirmativo reencaminha para o estado READ_FILE [Figura 4.22].

Se houver interrupções por parte do IPOS® durante o movimento, este estado reencaminha para o estado GET_REAL_POSITION, para se saber em que posição os motores pararam.

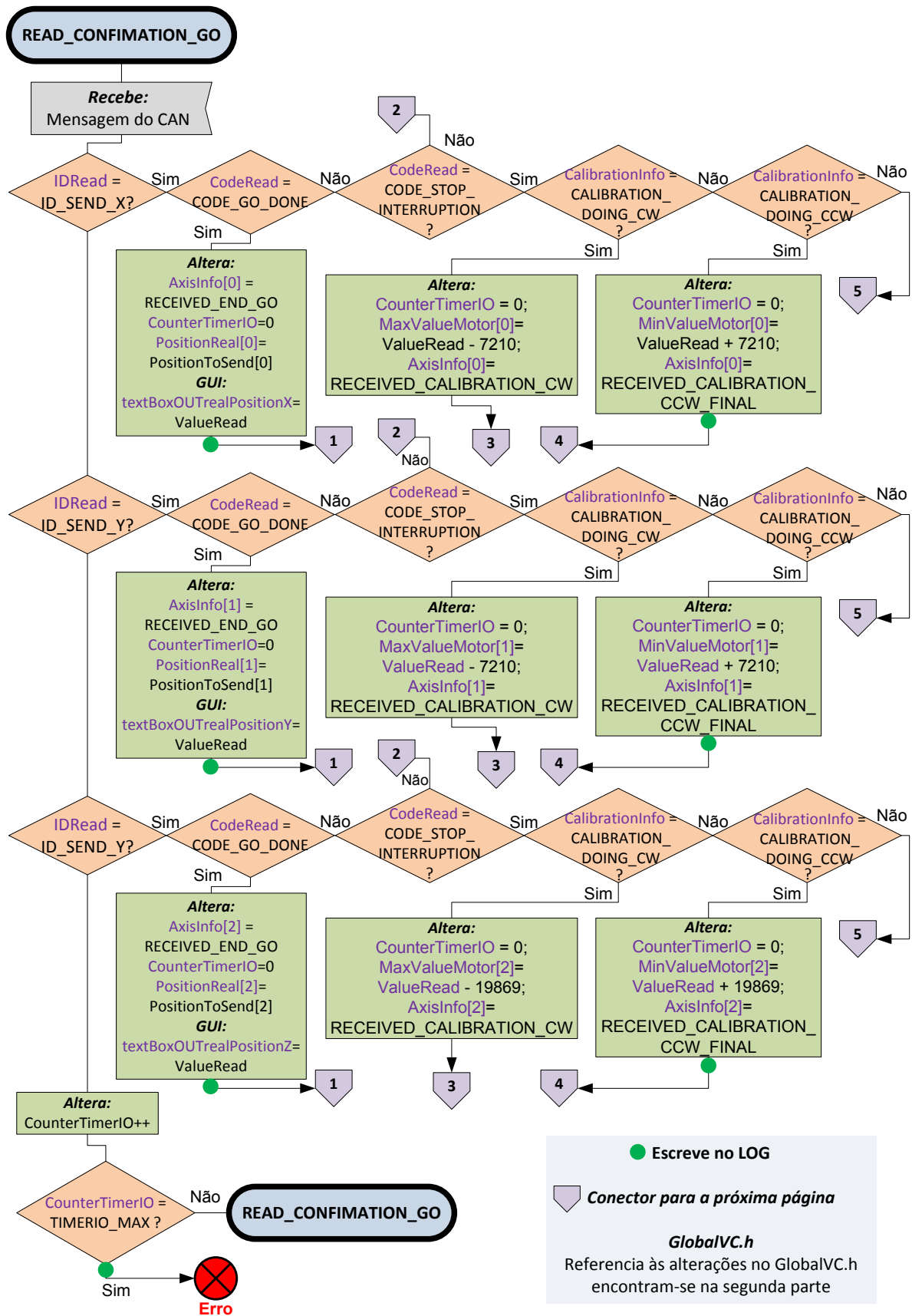



Figura 4.18: Diagrama SDL para o Estado READ_CONFIRMATION_GO – Continua na [Figura 4.19]

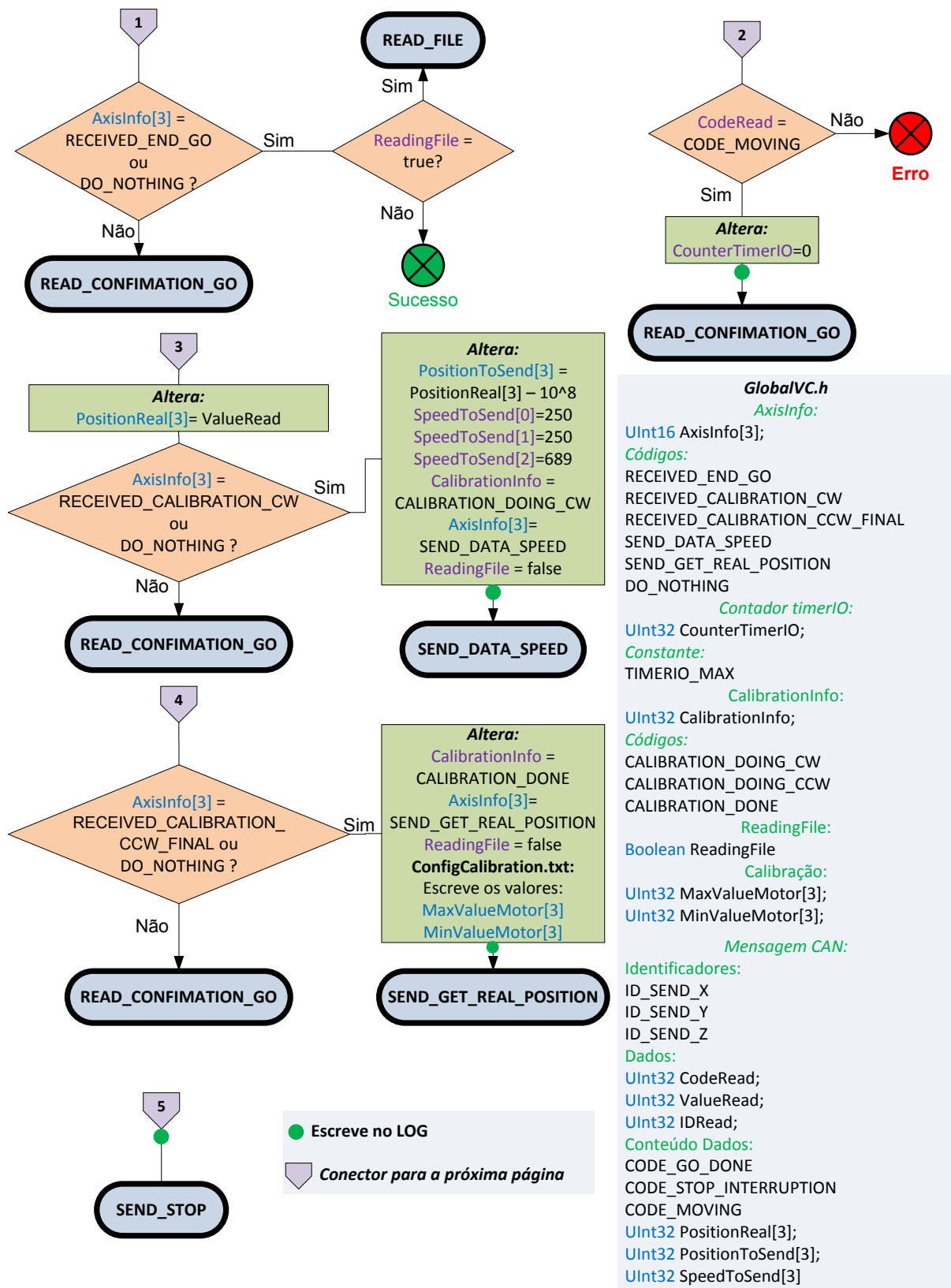



Figura 4.19: Continuação da [Figura 4.18] - Diagrama SDL para o Estado READ_CONFIRMATION_GO

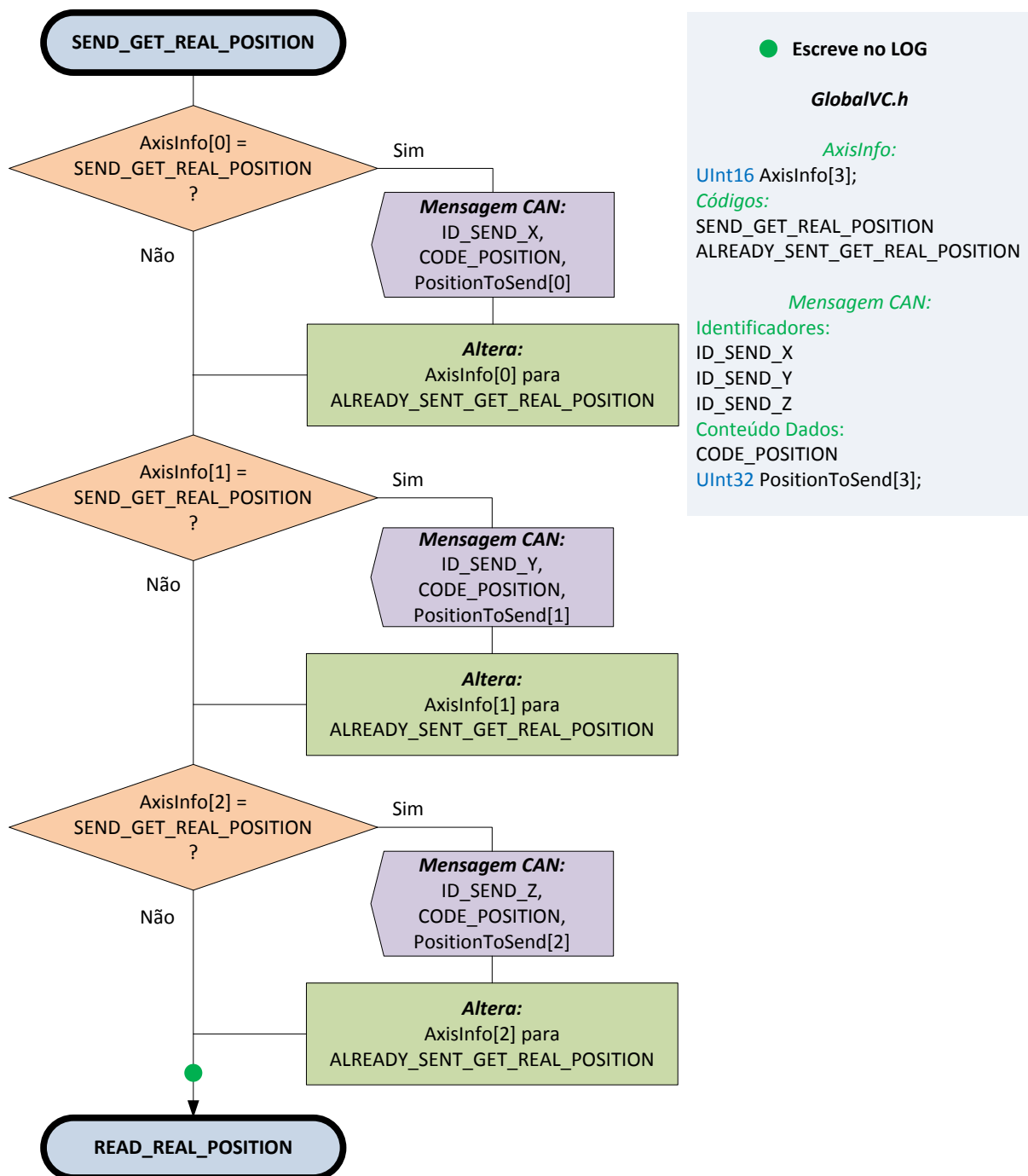



Figura 4.20: Diagrama SDL para o Estado SEND_GET_REAL_POSITION.

Terminada a sequência de estados discutida até aqui, pode-se passar para descrição do estado SEND_GET_REAL_POSITION [Figura 4.20], este estado também segue a mesma lógica do estado SEND_GO [Figura 4.17], mas em vez de enviar o código para iniciar o movimente envia o código a pedir as posições actuais dos encoders de cada motor. Terminado este pedido segue logo para o seguinte estado, o READ_REAL_POSITION [Figura 4.21].

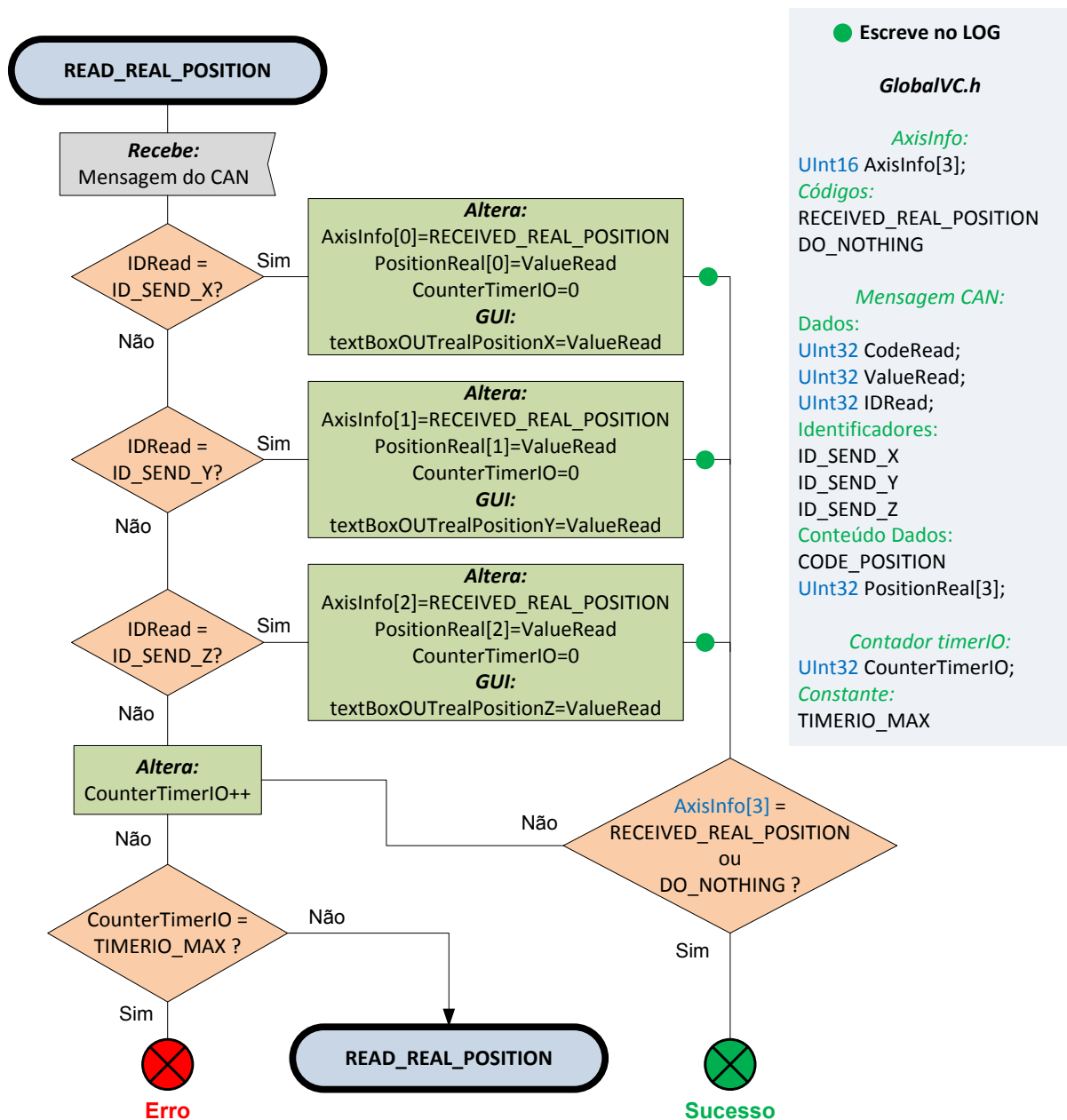



Figura 4.21: Diagrama SDL para o Estado READ_REAL_POSITION.

O estado READ_REAL_POSITION [Figura 4.21], sem fugir à regra dos estados iniciados com a palavra "read", lê o CAN-bus até receber as posições de todos os encoders para os quais enviou pedido, esperando apenas os 5 segundos entre as mensagens. E este estado fecha assim mais um grupo sequencial [Figura 4.12].

A abrir o próximo grupo temos o estado INITIALIZATION_FILE [Figura 4.23], este estado só activado uma única vez em cada leitura de ficheiros depois disso como já se viu o READ_CONFIRMATION_GO e que mantêm o processo de leitura ao longo do tempo. Assim, é neste

estado que verifica se os valores máximos e mínimos dos ficheiro e se é compatível com os tamanhos guardados da calibração, estes valores são também usados para ajustar os gráficos no OPMG.

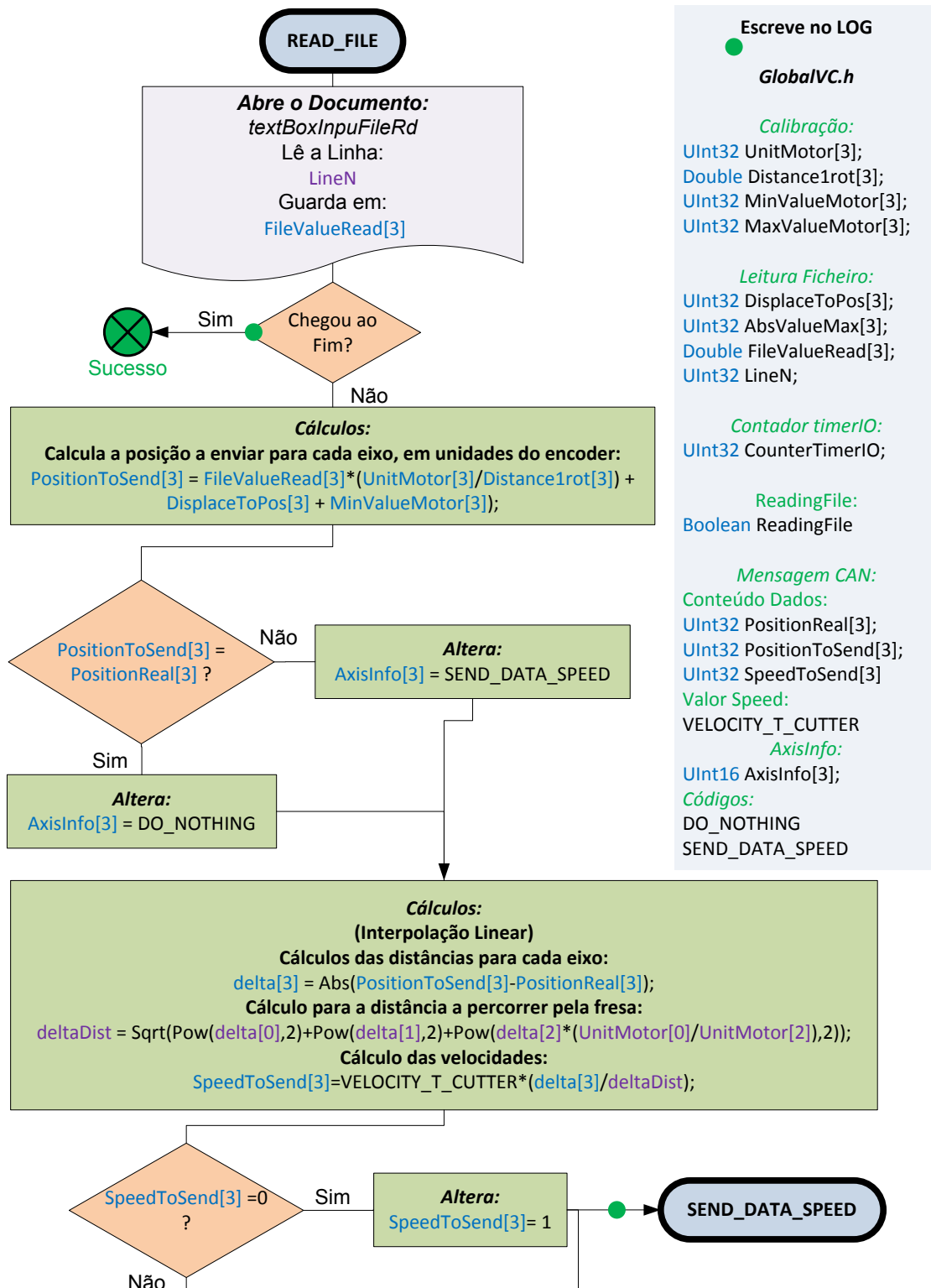



Figura 4.22: Diagrama SDL para o Estado READ_FILE.

INITIALIZATION_FILE

Abre o Documento:
textBoxInpuFileRd
Retira os valores máximos e mínimos de cada eixo.

Calculos:
Coloca os valores lidos em unidades compativeis com o encoder:
ValorMáximo = MáximoEixo*(UnitMotor[3]/Distance1rot[3]);
ValorMínimo = MínimoEixo*(UnitMotor[3]/Distance1rot[3]);
Calcula o valor a somar a cada valor lido para cada eixo, (para os colocar na zona positiva dos respectivos eixos).
DisplaceToPos[3] = Abs(ValorMínimo);
Calcula o maior valor possivel que se pode obter (já com os valores de calibração e transpostos para o eixo positivo) para cada eixo.
OPMG:
AbsValueMax[3]= ValorMáximo + DisplaceToPos[3] + MinValueMotor[3];

AbsValueMax[3] ≤ MaxValueMotor[3] ?

Não

Altera:
CounterTimerIO=0
ReadingFile=true

Erro

Sim

Altera:
CounterTimerIO=0
FileValueRead[3]=0

READ_FILE

Escreve no LOG

GlobalVC.h

Calibração:
UInt32 UnitMotor[3];
Double Distance1rot[3];
UInt32 MinValueMotor[3];
UInt32 MaxValueMotor[3];

Leitura Ficheiro:
UInt32 DisplaceToPos[3];
UInt32 AbsValueMax[3];
Double FileValueRead[3];

Contador timerIO:
UInt32 CounterTimerIO;

ReadingFile:
Boolean ReadingFile

Figura 4.23: Diagrama SDL para o Estado INITIALIZATION_FILE.

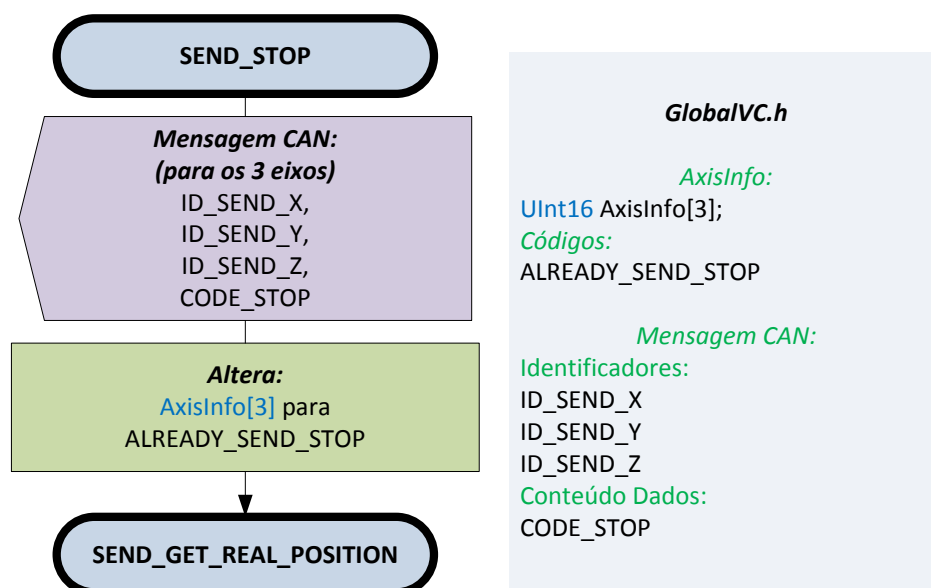



Figura 4.24: Diagrama SDL para o Estado SEND_STOP.

Depois de terminado os processos referentes ao estado INITIALIZATION_FILE [Figura 4.23], passa automaticamente para o estado READ_FILE [Figura 4.22], onde se vai ler linha à linha os valores para os guardar e depois calcular a interpolação linear para aplicar ás velocidades. Finalizando com a passagem para o estado SEND_DATA_SPEED [Figura 4.13], reiniciando assim todo um processo de comunicação e consequente movimento dos motores.

Para terminar esta descrição, fala-se do estado SEND_STOP [Figura 4.24], que é responsável por emitir ordens de paragem urgente de todos os motores. Depois de terminado este processo é reencaminhado para o estado SEND_GET_REAL_POSITION [Figura 4.20], para se saber em que posição ficou os encoders de cada motor depois desta ordem.

4.4 Evento do *timerSIM*

Se o objecto *radioButtonSimulation*, [Anexo A.4] estiver seleccionado então ao carregar um ficheiro CAM para leitura, este vai ser simulado em vez de começar a enviar os dados para os conversores inicializando assim o ciclo de leitura e movimento de motores, como já tinha sido dito.

Para correr esta simulação no tempo, foi criado um *timer*, de frequência determinado pelo objecto *textBoxSIMclock* [Anexo A.4] ao qual se deu o nome de *timerSIM*, e o seu modo de funcionamento está representado na Figura 4.25.

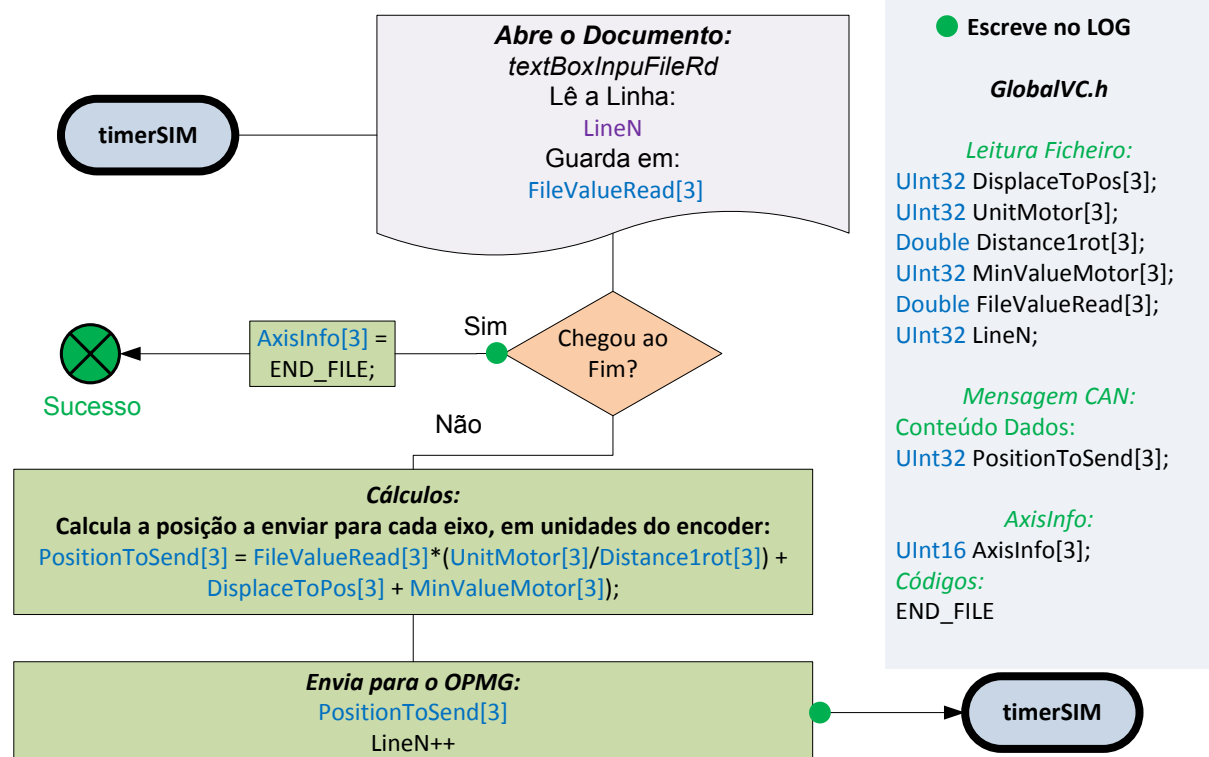



Figura 4.25: Evento para o timerSIM

4.5 Descrição dos códigos carregados nos controladores dos conversores

Do lado dos conversores de frequência, foi também necessário desenvolver códigos para que estes pudessem interpretar os dados enviados pelo DACS-OPMG através do CAN-bus. Assim sendo, criou-se também uma pseudo máquina de estados simples, que pudesse carregar os valores e dar ordem de movimento ou paragem dos motores. O código desenvolvido para cada conversor é praticamente igual aparte de uma linha que valida se o identificador associado à trama da mensagem do CAN-bus for o pretendido para esse nó. O código encontra-se representado por um diagrama na Figura 4.26.

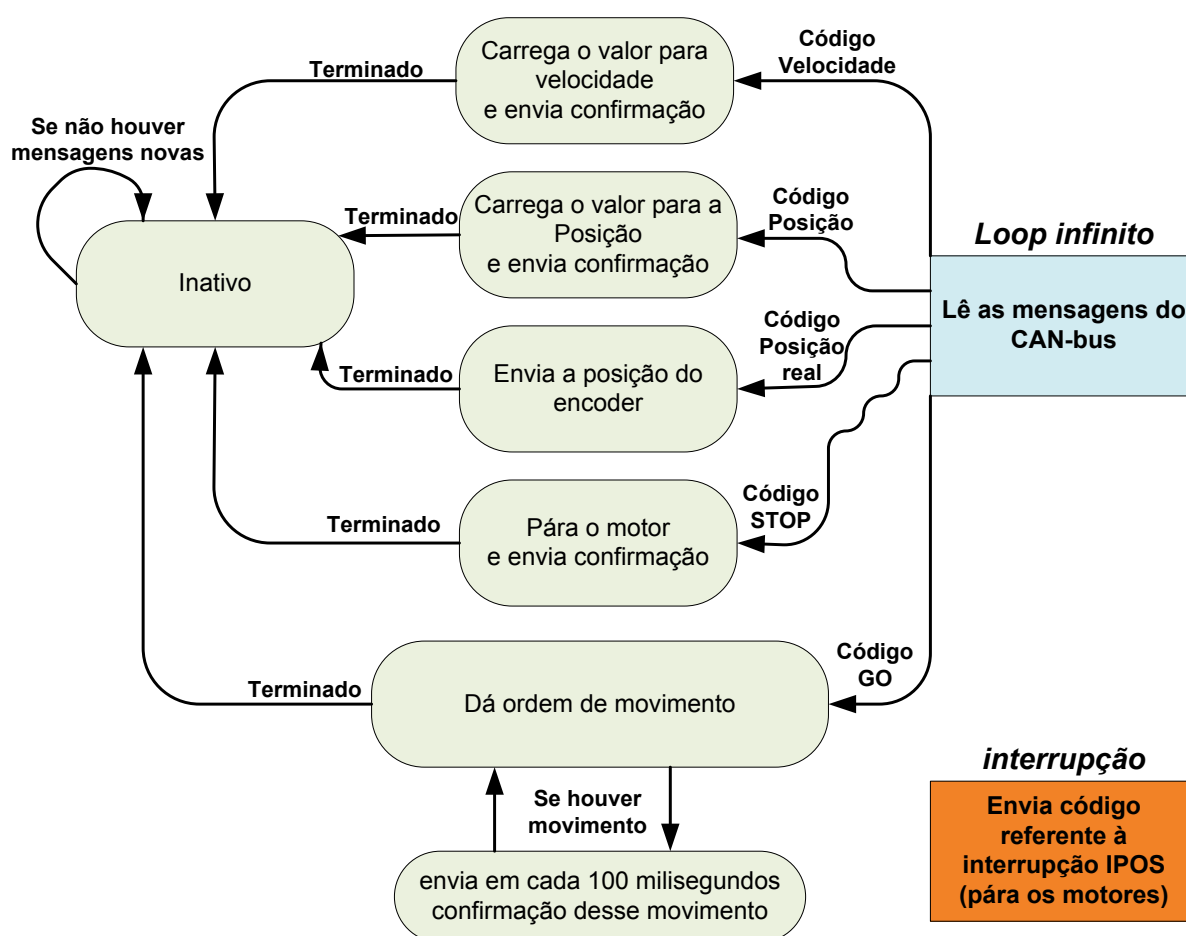



Figura 4.26: Diagrama explicativo do funcionamento dos códigos desenvolvidos para os conversores de frequência da [Sew]

Assim, o controlador do conversor está num *loop* infinito sempre verificando a rede CAN-bus. Se encontrar uma mensagem com o seu identificador, lê e conforme o código lido nessa mensagem, passa para o estado correspondente. Também se encontra representado a interrupção que é activa sempre que a ligação representada por “interrupção IPOS” na Figura 3.4, for activada.

5 Conclusões

Os objectivos propostos para o trabalho descrito nesta tese, foram alcançados em todos os aspectos apontados na introdução.

Assim, da realização do trabalho descrito nesta tese, surgiu um software-protótipo denominado DACS-OPMG, capaz de controlar os três moto-redutores da marca SEW-Eurodrive, através de conversores de frequência da mesma marca, de modo a que os movimentos gerados pelos moto-redutores descrevam trajectórias tridimensionais. O CAN-bus implementado através do produto CANUSB, que serve de comunicação entre o PC e os conversores de frequência. E finalmente, os programas desenvolvidos para os controladores dos conversores de frequência, foram capazes de interpretar as ordens recebidas do DACS-OPMG através do CAN-bus.

Está em curso o desenvolvimento do documento para a emissão de uma patente nacional sobre o sotware-protótipo DACS-OPMG.

Para visualização dos vídeos que demonstram o controlo dinâmico em funcionamento, aceder ao link [Videos].

5.1 Considerações sobre o DACS-OPMG

Assim como foi demonstrado no capítulo 4, através do DACS-OPMG é possível:

- Abrir/Fechar uma linha de comunicação com o CAN-bus através do CANUSB.
- Definir a velocidade de comunicação.
- Controlar as velocidades e posições de cada motor, quer individualmente, quer em grupo.
- Visualizar as posições actuais dos encoders.
- Efectuar uma calibração automática no eixo positivo referente a cada moto-redutor.
- Parar todos os motores em caso de emergência ou afins.
- Visualizar as projecções ortogonais das trajectórias descritas pela fresa.
- Simular ficheiros CAM, que cumpram os requisitos descritos no capítulo 2.5.
- Ler esses ficheiros CAM, e transpor essas coordenadas para movimento real, usando para tal, uma interpolação linear entre os pontos
- Definir as unidades equivalentes a uma volta completa em cada moto-redutor.
- Colocar em pausa todo o processo de movimento dos moto-redutores durante a leitura de um ficheiro CAM.
- Criar de hora em hora um ficheiro de diagnóstico.

5.2 Considerações sobre o CANUSB/CAN-bus

Como foi referido no capítulo 2.4, o dongle CANUSB, garantiu bons desempenhos para taxas de transferências menores ou iguais a $250kbps$, mas não se consegui transferir com taxas maiores que esta. Contudo os testes efectuados a esta velocidade cumpriram todos os requisitos impostos para este trabalho.

O buffer que o CANUSB possui foi mais que suficiente para o tipo de comunicação que se praticou, mas como também se pode ver pelos fluxogramas apresentados sempre que o DACS-OPMG iniciava mais um ciclo de comunicações com os conversores, limpava primeiro o buffer.

5.3 Considerações sobre o funcionamento dos códigos desenvolvidos para os conversores de frequência com controlo.

A nível de interpretação das variáveis fornecidas pelo IPOS relacionadas com o posicionamento e controlo sequencial dos motores, o código desenvolvido foi capaz de interpretar e fornecer os novos dados que recebia pelo CAN-bus.

5.4 Desenvolvimentos futuros

O DACS-OPMG foi criado de modo a facilitar no futuro a montagem dos moto-redutores na estrutura descrita no capítulo 1.1.1. O grupo “manual Control” do programa é uma boa ferramenta para controlar de forma precisa as posições dos motores.

Foi deixado em aberto as unidades de comprimento a aplicar para cada moto-redutor. Assim aquando a montagem, deve-se determinar qual a distância percorrida quando um moto redutor dá uma volta completa, e preencher esse valor para cada moto-redutor nas mesmas unidades que os ficheiros CAM.

Para garantir um funcionamento em conformidade com o que foi testado, os moto-redutores devem ser montados de modo a funcionar nos eixos positivos a que estão associados. E porque os encoders são incrementais, sempre que a alimentação a estes é cortada o encoder reinicia a posição para o valor zero. Por isso é conveniente ter os motores no “Home” quando se desligar se se quiser usar os mesmos valores de calibração guardados no ficheiro *ConfigCalibration.txt*.

Referências

[11358858PT] SEW; “*Redutores e motoredutores . Manual de instruções*”, 11358858/PT (2006)

[11535040PT] SEW; “*MOVIDRIVE compact MDF/MCV/MCS4_A - Instruções de Operação*” 11535040/PT (2007)

[Amaro, 2005] J. R. C. Amaro, R. L. A. Bettencourt; “*Concepção, projecto e fabrico de um digitalizador 3D*”. Instituto Superior Técnico (Outubro 2005)

[CANUSB] *www.canusb.com*, acedido em Outubro 2009

[Carvalho, 2004] R. Carvalho, S. Alves, P. Rosa, L. Alves; “*Concepção e Projecto de uma fresadora”*. Instituto Superior Técnico. (Outubro 2004)

[CIA] *www.can-cia.org*, acedido em Outubro 2009

[DOM756, 2007] L. Alves; P. Bicudo; P. Rosa; V.Cristino, *“Máquina Fresadora de estrutura modular para maquinagem de superfícies de grandes dimensões”,DOM756, (2007)*

[EN11320419] SEW; *“IPOSplus - Position and Sequence Control”*, EN11320419 (2004)

[EN16795210] SEW; “*DT/DV Gearmotors*” EN16795210 (2009)

[Franzoi] *www.franzoi.com*, acedido em Outubro 2009

[Hughes, 2006] A. Hughes; *“Electric Motors and Drives - Fundaments, Types and Applications”.* Pages **167-195.** Elsevier 3rd Edition (2006)

[Mesquita, 1997] R. M. D. Mesquita, J. M. C. Rodrigues, R. M. S. Batista; “*Máquinas-Ferramenta”*. Instituto Superior Técnico, Secção de Folhas 1ªEdição. (1997)

[Miller, 2004] R. Miller, M. R. Miller; *“Machine Shop Tools & Operations”*. Pages **233-241**. Wiley Publishing, Inc. 5th Edition (2004)

[Microsoft] www.microsoft.com/Express/VC/, acedido em Outubro 2009

[PT103652, 2007] L. Alves; P. Bicudo; P. Rosa; V.Cristino, *“Máquina Fresadora de estrutura modular para maquinagem de superfícies de grandes dimensões”,* PT103652, (2007)

[Richards, 2005] P. Richards; *"A CAN Physical Layer Discussion"* AN228. Microchip (September 2005)

[SEW] *www.sew-eurodrive.pt*, acedido em Outubro 2009

[Smid, 2003] P. Smid; “*CNC Programming Handbook*” Industrial Press Inc. 2nd Edition (2003)

[Suh, 2008] S. H. Suh, S. K. Kang, D. H. Chung, I. Stroud; “*Theory and Design of CNC Systems*”. Springer Series in Advanced Manufacturing. (2008)

[SurfIA] *www.surfinalentejo.com*, acedido em Outubro 2009

[Venda, 2003] P. Venda; “*Controller Area Network – Fundamentos*”. Instituto Superior Técnico (2003)

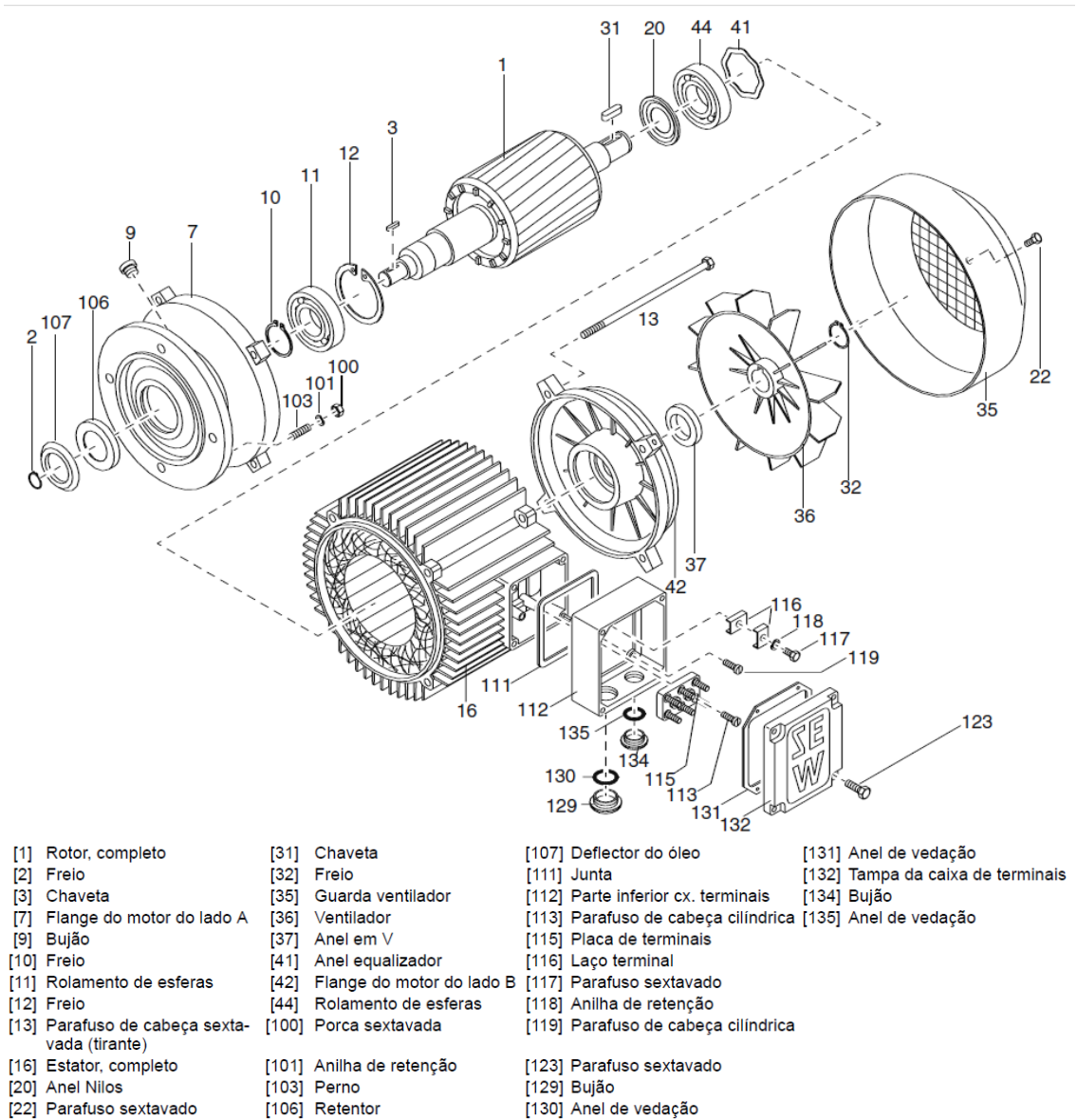
[Videos] *http://fisica.ist.utl.pt/~bicudo/VideosDACSOPMG.rar* criado em Novembro 2009

[Youssef, 2008] H. A. Youssef, H. El-Hofy; *"Machining Technology: Machine Tools and Operations"*. Pages **82-99**. CRC Press (2008).

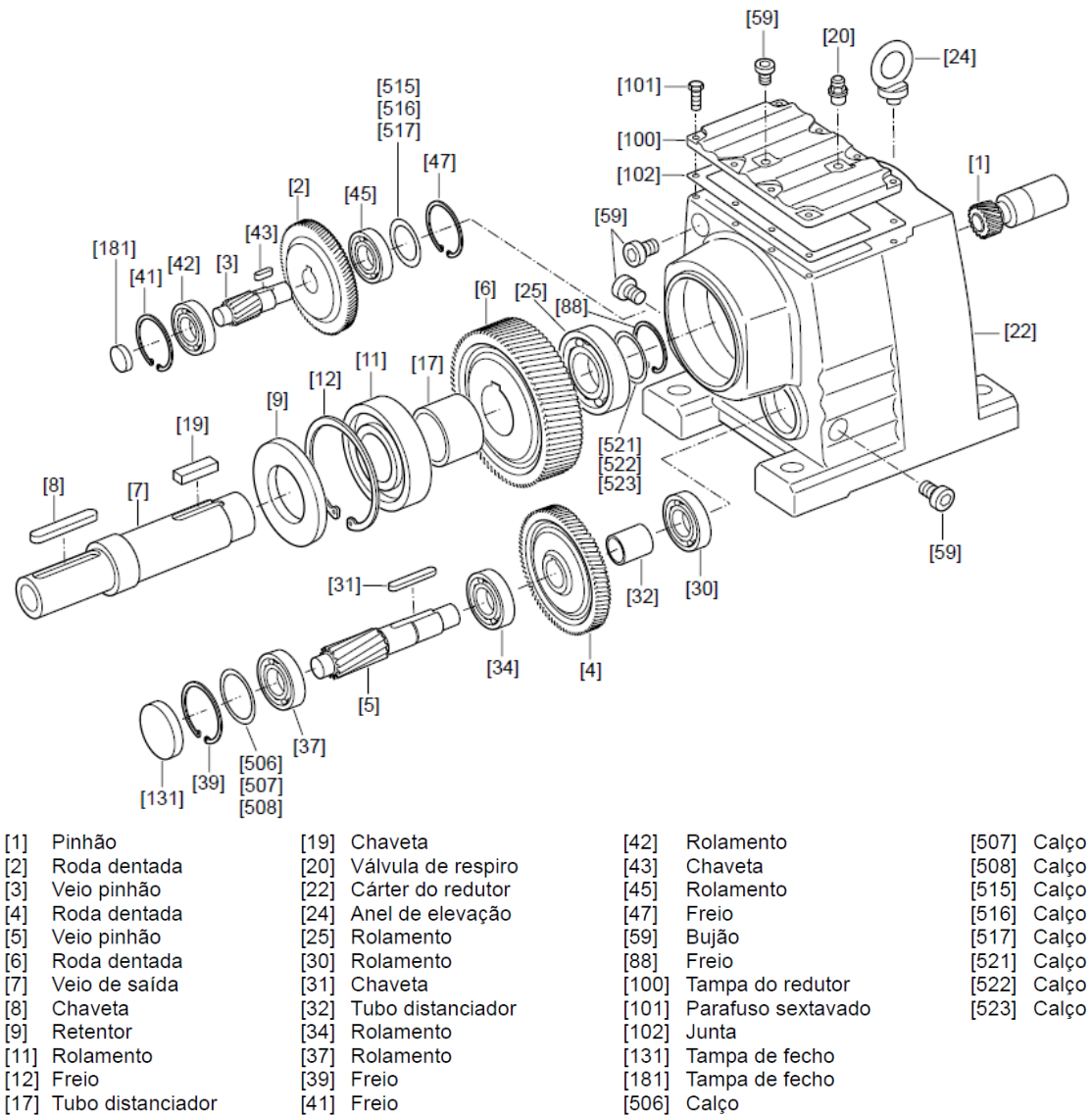
Anexos

A.1 Vistas explodidas [SEW]

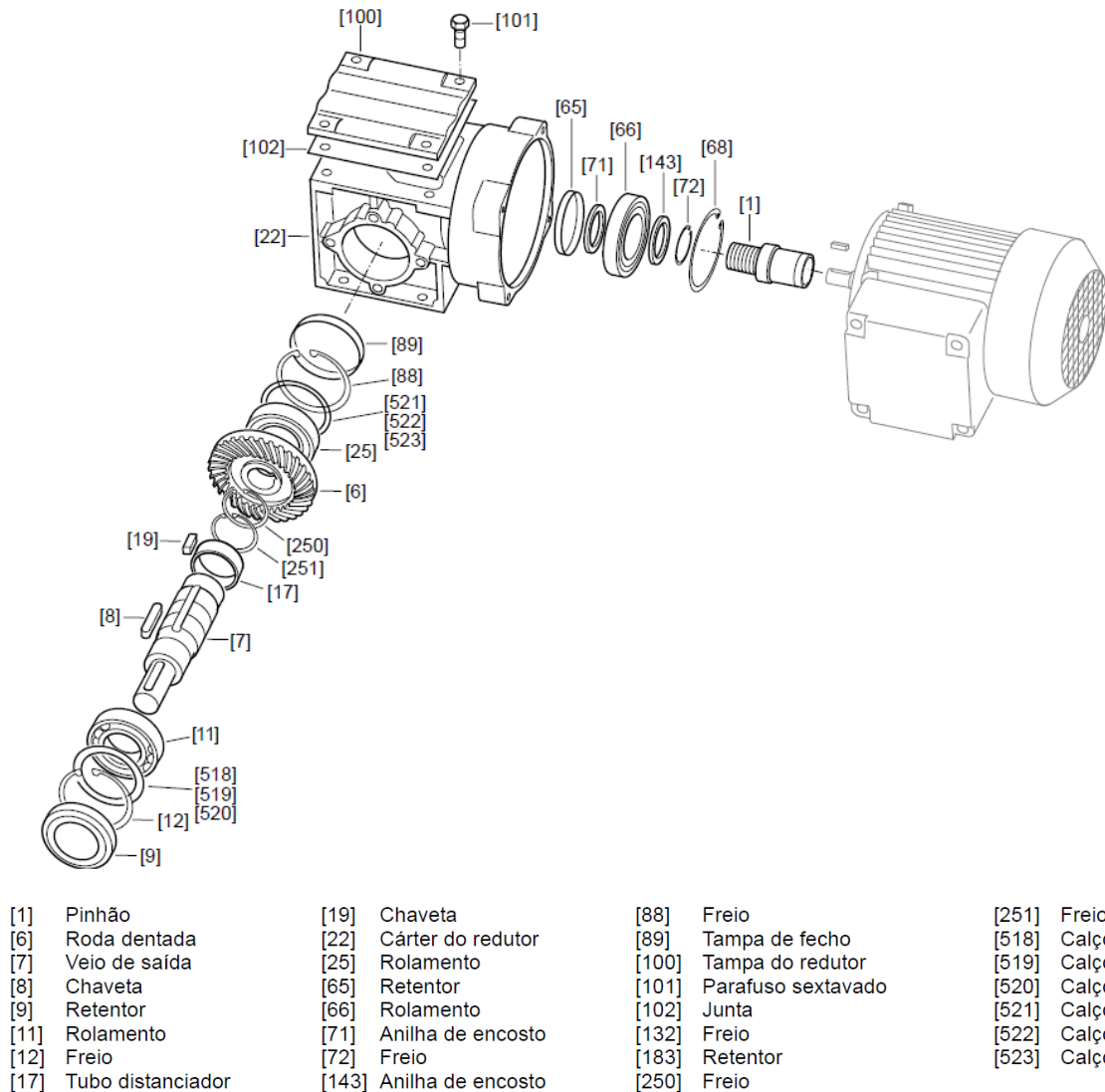
Motores DT e DR

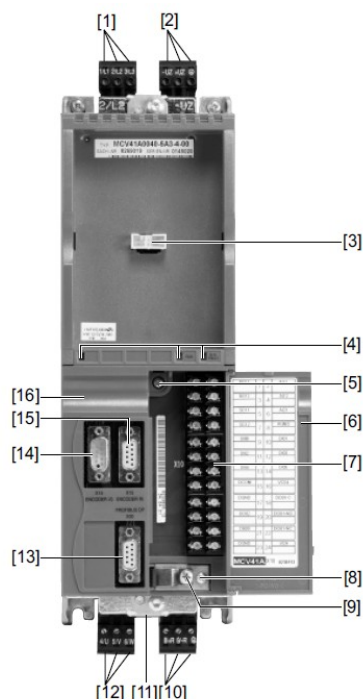
A [Figura_A 1] ilustra a estrutura geral dos motores [SEW] usados neste trabalho. Serve apenas para ter uma ideia da disposição das peças, não é uma réplica exacta.

Figura_A 1: Estrutura geral dos motores DT e DR (serve apenas para suporte de identificação de componentes) [EN16795210].

Redutores Helicoidais

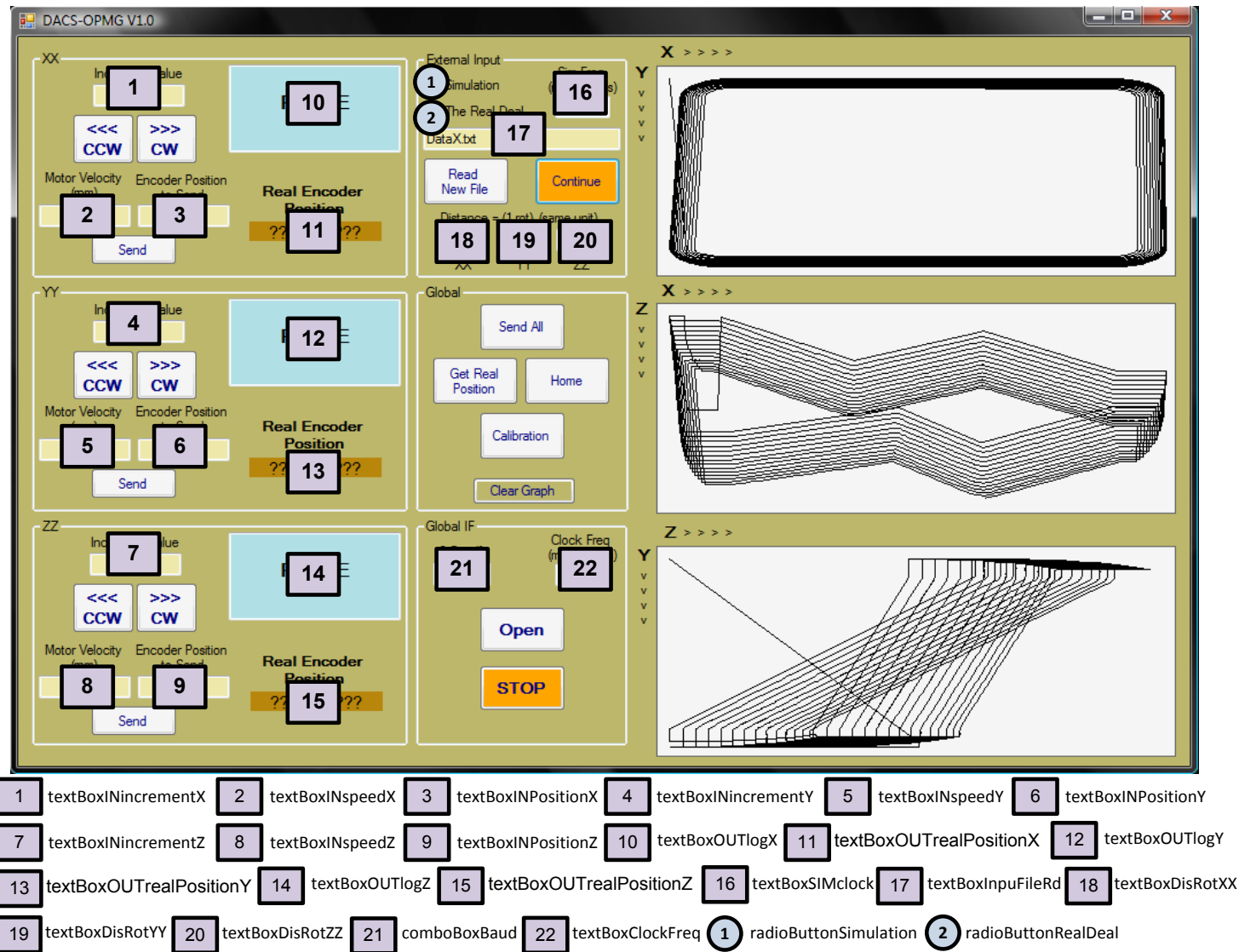
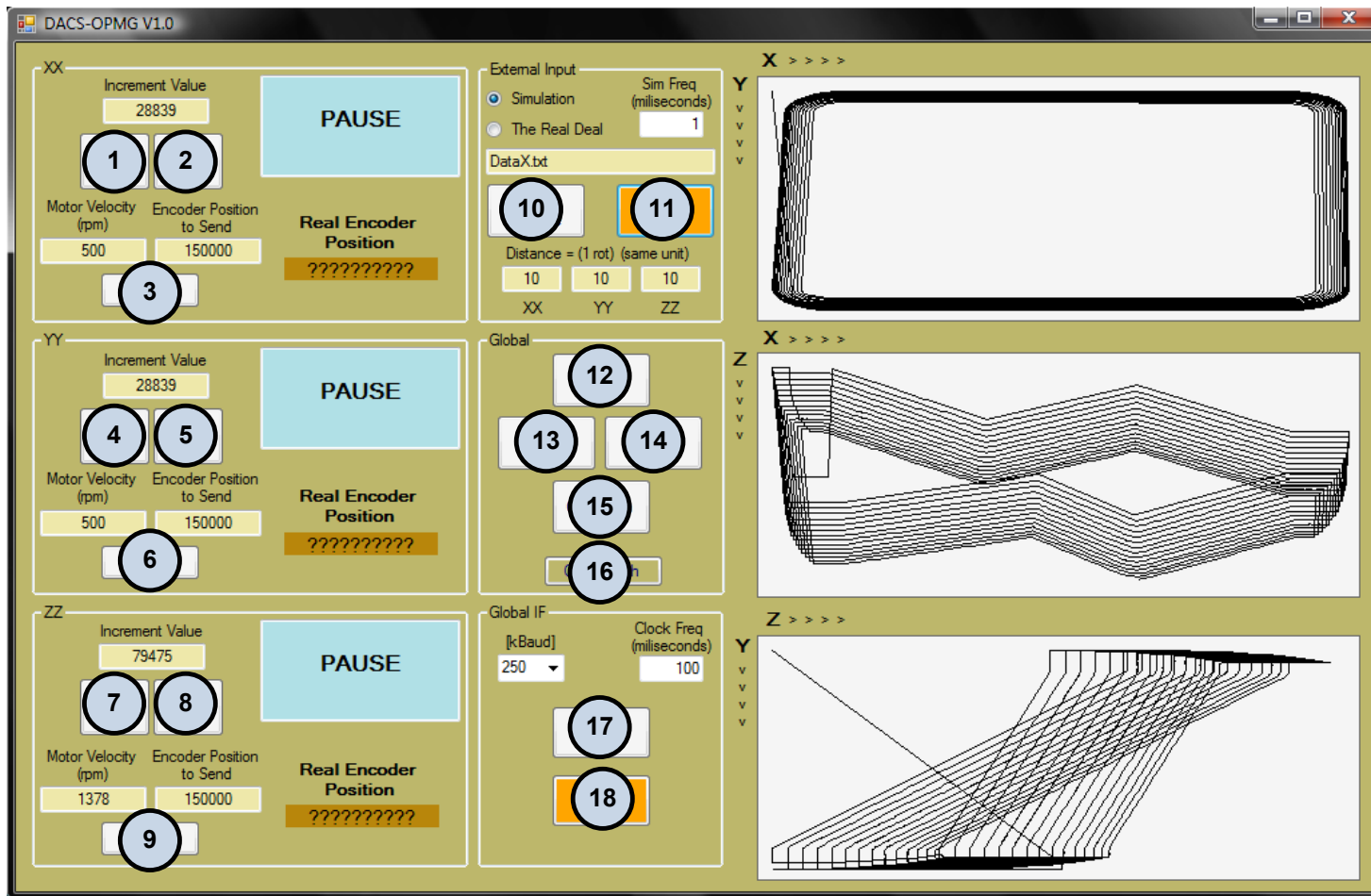
A [Figura_A 2] ilustra a estrutura geral dos redutores helicoidais [SEW] usados neste trabalho. Serve apenas para ter uma ideia da disposição das peças, não é uma réplica exacta.

Figura_A 2: Estrutura geral dos redutores helicoidais (serve apenas para suporte de identificação de componentes) [EN16795210].

Redutores Spiroplan

A [Figura_A 3] ilustra a estrutura geral dos redutores Spiroplan [SEW] usados neste trabalho. Serve apenas para ter uma ideia da disposição das peças, não é uma réplica exacta.

Figura_A 3: Estrutura geral dos redutores Spiroplan (serve apenas para suporte de identificação de componentes) [EN16795210].

A.2 Estrutura dos conversores de frequência com controlo.

[1] X1: Ligação da alimentação 1/L1, 2/L2, 3/L3, separável
[2] X4: Ligação do circuito intermédio $-U_Z/+U_Z$ e ligação de terra PE, separável
[3] TERMINAL: Slot para consola DBG ou opção USS21A/USB11A
[4] V1: LED de operação e LEDs de diagnóstico PROFIBUS (só para MCF/MCV/MCS41A)
[5] Parafuso de fixação A para a unidade de terminais
[6] Tampa da unidade de terminais com campos de anotação
[7] X10: Régua de terminais electrónicos
[8] Parafuso de fixação B para a unidade de terminais
[9] Parafuso para fixação do grampo de blindagem para a unidade de controlo
[10] X3: Ligação da resistência de frenagem 8/+R, 9/–R e ligação de terra PE, separável
[11] Ligação para o grampo da blindagem da secção de potência (não visível)
[12] X2: Ligação do motor 4/U, 5/V, 6/W
[13] Só para MCV/MCS41A X30: Entrada para PROFIBUS-DP (tomada Sub-D de 9 pólos)
[14] Só para MCV/MCS4_A X14: Saída para simulação de encoder incremental ou entrada para encoder externo (tomada Sub-D de 9 pólos)
[15] Só para MCV/MCS4_A X15: Entrada do encoder do motor (tomada Sub-D de 9 pólos)
[16] Unidade de terminais, removível

Figura_A 4: Estrutura dos conversores de frequência com controlo [11535040PT]

A.3 Identificação dos Objectos no GUI do Software-Protótipo DACS-OPMG.

Figura_A 5: Identificação dos objectos(Text Boxes, Radio Buttons e Combo Boxes) no GUI. "Para consultar durante a leitura do capítulo 4".

Figura_A 6: Identificação dos objectos(Buttons) no GUI. "Para consultar durante a leitura do capítulo 4".

A.4 Variáveis Globais (ficheiro GlobalVC.h).

Tabela_A 1: Descrição sobre as variáveis globais usadas (e definidas em GlobalVC.h) e como se enquadram na comunicação com o CAN-bus.

Varáveis e constantes	Descrição	
	Data-frame - CAN-bus	
int drvhandle	Guarda o valor do CANUSB referente à abertura de canal de comunicação	
	Variáveis de leitura do CAN-bus	
UInt32 CodeRead	Guarda o último código lido	4 primeiros bytes dos dados
UInt32 ValueRead	Guarda o último valor lido	4 últimos bytes dos dados
UInt32 IDRead	Guarda o último identificador lido	identificador
	Variáveis para enviar para o CAN-bus e manipulação interna (4 últimos bytes dos dados)	
UInt32 PositionReal[3]	Guarda a posição Real actual de [0]= XX, [1]= YY, [2]= ZZ	
UInt32 PositionToSend[3]	Guarda a posição para enviar para o CAN-bus em que: [0]= XX, [1]= YY, [2]= ZZ	
UInt32 SpeedToSend[3]	Guarda a velocidade para enviar para o CAN-bus em que: [0]= XX, [1]= YY, [2]= ZZ	
	[Constantes] Códigos de comunicação usados no CAN-bus (4 primeiros bytes do data-frame do CAN-bus)	
CODE_POSITION	Valor lido é uma posição	
CODE_REAL_POSITION	Pede a posição real do encoder	
CODE_GO	Coloca o motor em movimento	
CODE_GO_DONE	Motor terminou o movimento com sucesso	
CODE_STOP_INTERRUPTION	O interruptor IPOS activo	
CODE_STOP	Parar o motor	
CODE_SPEED	Valor lido é uma velocidade	
CODE_MOVING	O motor encontra-se em movimento	
	Ler e escrever em ficheiros	
Double FileValueRead[3]	Guarda os valores lidos no ficheiro CAM em que: [0]= XX, [1]= YY, [2]= ZZ	
UInt32 LineN	Contador para as linhas do ficheiro CAM	
UInt32 MinValueMotor[3]	Guarda o valor mínimo durante a calibração para escrever no ConfigCalibration.txt	
UInt32 MaxValueMotor[3]	Guarda o valor máximo durante a calibração para escrever no ConfigCalibration.txt	
UInt32 UnitMotor[3]	Guarda as relações entre os motores e redutores, em que: [0]= XX, [1]= YY, [2]= ZZ	
Double Distance1rot[3]	Guarda distancia que se obtêm para cada rotação, em que: [0]= XX, [1]= YY, [2]= ZZ	
UInt32 DisplaceToPos[3]	Guarda o valor a somar a todos os valores lidos para que estes estejam na zona positiva dos respectivos eixos, em que: [0]= XX, [1]= YY, [2]= ZZ	
UInt32 AbsValueMax[3]	Guarda o valor máximo (+DisplaceToPos[3]) para calibrar os gráficos do OPMG	
	Informadores	
UInt16 AxisInfo[3]	Informação sobre o estado actual dos eixos, em que [0]= XX, [1]= YY, [2]= ZZ	
Boolean ReadingFile	Informa se se está a ler um ficheiro CAM ou não: true=sim está a ler, não desativar o timerIO e false=não está a ler, o timerIO pode ser desactivado.	
UInt32 CounterTimerIO	Contador para o timerIO	
UInt32 CalibrationInfo	Informação sobre o ponto de calibração actual	